МОНИТОР

журнал для программистов — практиков

5.93

ISSN 0869-3889

PROGRAMMER

Работа над ошибками

Borland, ты неправ!

Интервенция в страну DOS

HARDWARE

Модемы — Ваши, линии связи — наши

GAMES

Джафар — попытка реванша

19200 BPS
Сертификат Минсвязи РФ
№ ОС-1-ТМ-41

БЫСТРО. ПРОСТО. НАДЕЖНО.

Ваши потребности уже выросли за пределы нынешнего модема. Вам нужен факс–модем, который действительно отвечает современным стандартам. В самом деле выбор предельно простой. Только ZyXEL предлагает Вам универсальные, высокоскоростные факс–модемы с голосовыми возможностями: успешно проверенные на российских телефонных линиях, с квалифицированной поддержкой, по цене, которая не пугает. Ведь телефонное время тоже стоит денег...

Официальный дистрибьютер
А/О МКЦ Вариант
117279 Москва, ул. Островитянова, 37а
Тел. 932–7601, 420–2519
Факс 932–7410, 420–5311

Круглосуточно:
Голос/Факс демо 932–8510
Информация
max@mechta.msk.su2:5020/22@fidonet
WHITE BEAR BBS 932-8465!

Есть ли на свете человек, который мог бы обнять необъятное?
Козьма Прутков
Мысли и афоризмы, № 91а

Специалист или коллектив специалистов, перед которым поставлена задача разработать сложную компьютерную систему, неоднократно попадает в ситуацию, когда необходимо сделать выбор между различными направлениями дальнейшего развития проекта.

Должен ли я разрабатывать систему только на IBM PC, объединенных в локальную сеть? Или, может быть, в качестве мощных серверов установить две машины AS/400? Устраивают ли меня при моих требованиях к надежности работы системы компьютеры марки Лэнд? А компьютеры Dell? Будут ли мои программы нормально работать на 386-х с 4 Мбайтами оперативной памяти или необходимы 486-е с 16 Мбайтами и винчестером с емкостью гигабайт? И хватит ли мне при покупке аппаратуры выделенных на это денег?

Какую СУБД следует выбрать для разработки: Clipper, Clarion, FoxPro? А может быть, Informix или Oracle? Способна ли эта СУБД нормально общаться с прикладными программами на языке C, откомпилированными с помощью компилятора Microsoft? А как насчет Zortech C++ или Watcom C? Какую выбрать библиотеку функций для работы с графикой?

Какую аппаратуру и программное обеспечение я должен приобрести для эффективной организации работы офиса? Лексикон или Word Perfect? Lotus или QuattroPro? Или то же самое, но под Windows? Выбрать модемы, коммуникационные пакеты, средства обеспечения безопасности...

Каждый раз подобный выбор является весьма ответственным, так как влечет за собой значительные финансовые и временные затраты, а ошибка может привести к провалу проекта. Для того чтобы сделать правильный выбор, необходима подробная и точная сравнительная информация по каждой группе программных или аппаратных средств, сопоставимых по своему назначению и возможностям.

Подготовка и публикация таких обзоров, предназначенных для программистов, в «Мониторе», для более широкого круга работающих с компьютерами людей — в «Монитор-Аспекте» — это одна из основных задач нашей редакции. В качестве примеров можно привести обзор А. Сидорова, сравнивающий оптимизацию в разных компиляторах C++ («Монитор» 8.92), обзоры Е. Равича по модемам в этом номере «Монитора» и по компьютерным сетям («Монитор-Аспект» 1.93), В. Барсукова по системам обеспечения безопасности связи («Монитор-Аспект» 1.93). Редакция и наши постоянные авторы будут продолжать подготовку таких обзоров и далее.

Однако...

Опять скажу: никто не обнимет необъятного!
Козьма Прутков
Мысли и афоризмы, № 169

Качественная подготовка любого такого сравнительного обзора требует привлечения к работе над ним нескольких специалистов, имеющих опыт работы с каждой из рассматриваемых систем, способных создать методику испытаний, протестировать системы, проанализировать их работу с разных точек зрения, аргументированно и понятно изложить результаты.

Мы приглашаем квалифицированных специалистов, как программистов, так и электронщиков, с опытом работы с самыми разными программными и аппаратными средствами, желающих пополнить свои профессиональные знания и навыки и готовых уделить часть своего свободного времени работе в тестово-аналитической лаборатории журнала «Монитор».

Требования: высокая квалификация, логический ум и способность к анализу и ясному изложению своих мыслей на бумаге, энтузиазм, выносливость, чувство юмора.

На высокую оплату рассчитывать не стоит (по крайней мере, поначалу). Зато гарантируем возможность доступа и экспериментирования с разнообразными программными и аппаратными средствами, компьютер и диван на ночь.

Вы можете написать нам по адресу: 117419 Москва, а/я 765 или позвонить по телефонам 238-99-33 и 335-68-31 (вечером).

Плюнь тому в глаза, кто скажет, что можно обнять необъятное!
Козьма Прутков
Мысли и афоризмы. № 104

Господа программисты, электронщики и бизнесмены!

Если вы считаете, что ваша программа или аппаратура конкурентоспособны и хотите, чтобы она была включена в публикуемые на страницах «Монитора» или «Монитор-Аспекта» обзоры;

если вы хотите провести независимое объективное бета-тестирование вашей системы;

если вы хотите усовершенствовать документацию и демонстрационные ролики;

наконец, если вам нужно содействие в изготовлении документации и коробок для ваших систем, —

во всех этих случаях вам стоит написать и/или позвонить нам.

Редакция журнала «Монитор»

Журнал для программистов-практиков

Редколлегия:
Д.Э.Бройтман
(главный редактор)
О.А.Балдина
В.В.Водолазский
В.С.Головин
Н.Н.Косарев
А.М.Миляков
(ответственный секретарь)
Н.А.Никитина
(отдел рекламы)
А.В.Орехов
(издательский директор)
Е.В.Равич
Л.И.Турок

Графическое бюро:
С.В.Макельский
(главный художник)
Г.Л.Карчикян
(компьютерная верстка)

Отдел распространения:
Р.А.Болдырев
А.А.Винокуров
Е.В.Гун

В оформлении номера использованы рисунки
А.С.Замулы

Подписано в печать с оригинал-макета 09.07.93
Формат 60x84/8.
Тираж 30 000 экз.
Печать офсетная. Печ. л. 10,0
Цена договорная
Зак. 3859
Отпечатано в типографии ИПК «Московская правда»

Издатель:
Научно-техническое общество

Софт · Москва

Для писем: 117419 Москва, а/я 765
Тел. (095) 238-99-33
Факс (095) 238-32-78
Для справок (095) 943-25-53
Телекс 613322 GRANT SU
BBS (095) 135-14-39 ARSORBIS

© "Монитор", 1993

Содержание

Письма

А. Аброськин. Еще один вариант программы распечатки каталогов 4

Д. Правиков. Еще раз о структуре PSP 4

Технологии программирования

С. Курковский. Гипертексты с практической точки зрения. Часть 2 6

В статье рассказывается об основах организации гипертекстовых систем. Основное внимание уделено проблемам построения и применения гипертекстов. Предполагается опыт работы на C++. Вторая часть посвящена вопросу создания гипертекстового интерпретатора.

Д. Арапов. Работа над ошибками 12

Систематизируется опыт обработки нештатных ситуаций на Turbo Pascal. Произведен анализ нескоторых механизмов обработки исключительных ситуаций, а также приведен текст программы, реализующей такой механизм.

В. Захаров, И. Кузнецов. Turbo Vision: обнаружение некоторых ошибок в программах 20

В статье рассматриваются практические приемы поиска некоторых "скрытых" ошибок в программах, создаваемых в среде Turbo Pascal 6.0 и Turbo Vision. Приведено описание некорректной работы модуля HelpFile из библиотеки Turbo Vision и даны рекомендации по устранению обнаруженных ошибок.

Аппаратура

Д. Бройтман. Микропроцессор Pentium. Часть 3 26

Процессор Pentium включает в себя мощную подсистему поддержки вычислений с плавающей точкой и с целыми числами большой размерности.

DOS под микроскопом

Д. Валдин. Резидентные программы на языке C. Часть 1 32

Статья посвящена проблемам написания резидентных программ на языке C. Описываются модели данных и особенности проектирования для компиляторов фирм Borland и Microsoft.

Мир графики

А. Тихонов. **Показать картинку. Часть 2** 44

В статье расмотрен графический формат GIF фирмы Compuserve, приводится программа визуализации графических образов в формате GIF на Turbo Pascal.

Языки программирования

С. Позднеев. **Применение языка Ада для разработки ПО ПЭВМ** 52

Рассматривается и обосновывается применение алгоритмического языка Ада и соответствующей технологии программирования для разработки программных систем на ПЭВМ. Приводятся сравнительные характеристики наиболее универсальных и распространенных систем программирования на языке Ада.

Коммуникации

Е. Равич. **От 2400 до 19200: не ругайте наши линии связи** 60

САПР

А. Фрумкин, Р. Глуховский. **Система разработки приложений в Автокаде** 66

Игры

В. Водолазкий. **Убить Джафара нелегко!** 72

Статья раскрывает дотошному читателю ужасные и кровавые тайны второй части игры "Prince of Percia".

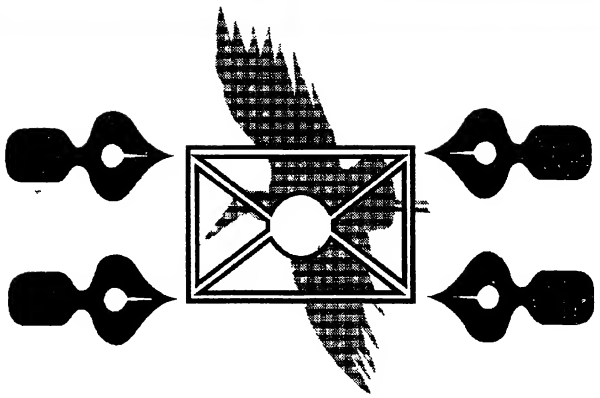
Д. Чикин. **Пристальный взгляд на компьютерные игры** 74

Точка зрения программиста на процесс разработки и продажи компьютерных игр.

Всякая всячина

Л. Ливеровский. **SIG PARADOX** 78

В статье рассказывается о работе пользователей бизнес-систем фирмы Borland и о сравнении скоростных характеристик PARADOX 4.0 и 3.5, которое было выполнено членами этой группы.



Еще один вариант программы распечатки каталогов

Редакция с удовольствием отмечает, что журнальные публикации вызывают горячий отклик наших читателей. А.Аброськин из Нижнего Новгорода предложил новый вариант программы распечатки текущих каталогов (см. «Монитор» № 3.92), отличающейся

- малым размером (90 байтов);
- большей скоростью работы.

Может быть, Вы сумеете перекрыть этот рекорд?

```
code    segment
        assume  cs:code, ds:code, es:code
        org     100h

start:
        mov     ah,30h              ; get ver.DOS
        int     21h
        cmp     al,4
        jae     ml
        mov     cs:[records],51h    ;if < 4.x
ml:     mov     ah,52h              ;
        int     21h                 ;"Get List of Lists"
        mov     al,es:[bx+20h]      ;
        mov     cs:[drives],al      ; количество дисков в системе
        mov     di,es:[bx+16h]      ;
        mov     ax,es:[bx+18h]      ; <ES,DI> PTR to CDs
        push    ax
        pop     es

m2:     push    es
        push    di
        mov     ax,1212h
        int     2Fh
        push    cx
        xor     bh,bh
        mov     ah,03h
        int     10h
        pop     cx
        mov     bx,0009h
        mov     ax,1301h
        push    di
        pop     bp
        int     10h
                                    ;
        mov     ax,0E0Dh            ;
        int     10h                 ;
        mov     AX,0E0Ah            ;
        int     10h                 ; Press <ENTER>
                                    ; до полного
                                    ; удовлетворения
        pop     di                  ; (в смысле LOOPNZ)
        pop     es                  ;
        add     di,cs:records       ;
        dec     byte ptr cs:drives  ;
        loopnz  m2                  ;
                                    ; ATTENTION:
        ret                         ; thats ALL..LL..LL..L..
;-------------------------------------------------------------
drives db       ?                   ;
records dw      58h                 ; only for 4.x
code    ends
        end     start
```

■

Еще раз о структуре PSP

Вы никогда не интересовались, зачем люди пишут статьи? Одни для того, чтобы рассказать о новых идеях, другие — чтобы подтвердить или опровергнуть их, третьи обобщают все написанное в обзорах. Разумеется, помимо всего перечисленного существуют еще и другие причины. Например, в нашей стране, в силу ограниченного доступа к необходимой информации, такой причиной является стремление систематизировать и донести до тех, кому это недоступно, техническую документацию. Труд авторов, пишущих такие статьи, можно сравнивать с трудом просветителей, но только в том случае, если они делают это честно.

Поводом для написания данной статьи явилась публикация Г.Н.Никольского и Д.В.Гуля ([1]), рассматривающая некоторые управляющие структуры DOS и, в частности, PSP — префикс программного сегмента. Приведенные в статье комментарии к отдельным полям PSP даны настолько туманно, что способны поставить в тупик даже профессионала. Более того, они выполнены в таком стиле, как будто основное содержание статьи посвящено проблемам не программирования, а тренировки сообразительности. «Теперь почти понятно...», «становится понятно...» — так, например, начинаются комментарии к отдельным полям PSP. Судя по всему, авторы считают тему исчерпанной, поэтому приходится за них доводить работу до конца. Из-за того что данная статья является ответом на [1], полная структура PSP здесь не приводится.

Итак, начнем с первого поля. Смещение в таблице 0h, размер — 2 байта. В [1] указано, что это CP/M выход и что, организовав переход на это поле с помощью команды jmp, можно вернуться в DOS. «Однако использовать это нежелательно, в DOS лучше выходить через функцию 4Ch int 21h». И все. Понимайте, как хотите. Давайте попытаемся расшифровать, что стоит за этой загадочной фразой.

Данное поле для любых программ всегда содержит одинаковое значение — CDh, 20h, или, переведя этот код на язык ассемблера, команду int 20h. Это прерывание, функционально идентичное функции 0h int 21h, завершает выполнение текущей задачи. Единственное требование при вызове обоих прерываний заключается в том, что регистр CS должен содержать сегментный адрес PSP. Любая COM программа практически всегда удовлетворяет этому условию, а вот EXE — нет. Именно поэтому для нее рекомендуется использовать функцию 4Ch int 21h, которая, помимо того, что избавляет от формирования значения CS, позволяет передать командному процессору код завершения. Если помните, то на коде завершения основано ветвление в командных файлах — if errorlevel и так далее.

Что же касается передачи управления на это поле с помощью команды jmp, то попробуйте ответить на вопрос: каков минимальный объем COM программы, правильно возвращающей управление в DOS? Два байта, то есть int 20h? А вот

и нет. Такая программа имеет размер 1 байт и состоит из одной команды ret! Дело в том, что при загрузке программы в память DOS записывает на дно стека значение 00h. При выполнении команды ret оно загружается в IP, и управление таким образом попадает на команду int 20h в PSP. Тем не менее, никто не может запретить Вам завершить EXE программу командой ret. Но только в этом случае Вы должны сами позаботиться о том, чтобы стек содержал необходимые значения. Тогда начало программы может быть оформлено так:

```
push ds
xor  ax,ax
push ax
```

Обратите внимание на то, что в стек помещаются два слова для обеспечения межсегментного перехода. Соответственно завершающей командой для EXE программ будет retf.

Кстати, если авторы упоминали CP/M, то почему они не пояснили, при чем здесь другая операционная система? Насколько мне известно, первая версия DOS выпускалась как временная замена CP/M для 16-разрядных машин, поэтому считалось, что она должна быть совместима с ней как по системным вызовам, так и по основным управляющим структурам данных.

Поле 4 — far call в int 21h. Согласен. Но дальше идет фраза, с которой согласиться никак нельзя: «при его использовании, если повезет, Вы сэкономите несколько микросекунд». И вот почему. Как Вы понимаете, вызывать функции DOS, осуществляя переход на это поле с помощью команды jmp, нельзя, так как Вы никогда не получите управление обратно. В отличие от поля 15, здесь нет команды retf. Поэтому единственным вариантом его использования остается косвенный вызов межсегментной процедуры, причем, поскольку DOS возвращает управление командой iret, для сохранения сбалансированности перед вызовом в стек необходимо помещать флаги. Например, пусть ES содержит сегментный адрес PSP, а BX — смещение в нем значения вектора 21h, то есть 6. Тогда обращение к DOS должно осуществляться с помощью следующего набора команд:

```
pushf
call  far es:[bx]
```

Теперь давайте считать время выполнения этих команд. Чтобы обеспечить независимость от тактовой частоты конкретного компьютера, считать будем в тактах процессора. Команда pushf занимает 10 тактов. Команда call far — 37 тактов плюс время на вычисление эффективного адреса, которое в данном случае минимально — 5 тактов. Итого 52 такта. Теперь сравним это значение с 51 тактом, необходимым для выполнения int 21h, и попытаемся ответить на вопрос, как же должно везти человеку, чтобы он сэкономил таким образом несколько микросекунд?

В принципе, можно обмануть DOS, записав в следующее поле (номер 5, смещение 0Ah) команду retf. Но в этом случае перед завершением программы Вы должны восстановить прежнее значение этого поля. Понимаю, что, имея перед собой только [1], вряд ли можно понять мой настойчивый совет, поэтому поясняю. Поле 5 хранит адрес, по которому будет передано управление при завершении текущей программы. То, что этот адрес совпадает со значением вектора 22h, представляет собой частный случай запуска программы, когда родительской программой является командный процессор (как правило, command.com). В общем случае эти значения могут не совпадать, и DOS при завершении задачи ориентируется только на значение поля 5 PSP.

Следующее замечание касается описания полей 9, 12 и 13, которые содержат локальную таблицу указателей открытых файлов (handle), ее адрес и размер. В целом значения этих полей описаны верно, неверным является трактовка параметра FILES в config.sys. Исходя из [1], можно понять, что, если в память компьютера одновременно загружены несколько программ и каждая программа, открывая файлы, использует только локальную таблицу указателей файлов, то в config.sys достаточно записать FILES=20. Это глубоко ошибочное убеждение. Параметр FILES ограничивает число файлов, одновременно открытых в системе. Это значит, что, если программа A открыла 20 файлов (считая стандартные устройства ввода-вывода), а затем передала управление программе B, которая также открыла 20 файлов, то в сумме обе программы открыли, скажем, 30 файлов (часть файлов может быть одновременно открыта для двух программ). Но если параметр FILES ограничен значением 20, то программы A и B работать одновременно не смогут.

Поля 20 и 21, помимо хранения остатка командной строки, используются как область обмена данными с диском (DTA — data transferring area). Поэтому, если Вы не переназначали в программе эту область в другое место и до разбора командной строки использовали такие функции, как findfirst и findnext, то можете уже ни на что не надеяться. Остаток командной строки будет безвозвратно потерян.

Последнее замечание касается программы, разбирающей структуру PSP. Если авторам не нравятся утилиты типа memlook, то это их личное дело. Но при написании собственных утилит не надо популяризировать свои заблуждения. Примером такого заблуждения может служить разбор параметров окружения. Как известно, DOS, начиная с версии 3.0, за концом окружения помещает строку, содержащую путь загрузки программы. Однако в приведенном листинге вывод параметров окружения написан таким образом, как будто самой старшей версией DOS на сегодняшний день является 2.0.

И еще. Во всей первой части публикации (вторая часть выйдет в следующем выпуске БИТ) нет ни одной ссылки на источник информации, из которого брались структуры управляющих таблиц DOS. Как Вы понимаете, в этой ситуации можно сделать два вывода. Либо авторы самостоятельно исследовали «начинку» DOS, либо опубликовали чей-то плохой перевод. Какой вывод более вероятен, судить вам. Моим же основным источником информации является [2], который и рекомендую использовать всем. ■

Литература

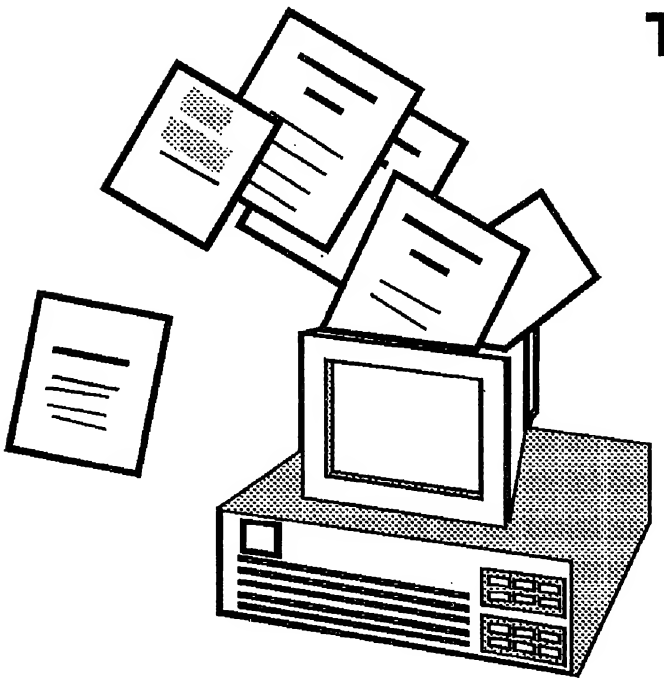
1. Г.Н.Никольский, Д.В.Гуля. Таблицы DOS.//Библиотека информационной технологии; Под ред. Г.Р.Громова. – М.: ИнфоАрт, 1992. – Вып. 6.
2. Dawe Williams. Programmer's Technical Reference for MS-DOS and the IBM PC.– 1989.

Д. Правиков

Гипертексты с практической точки зрения.

С. Курковский

Часть 2



В первой части настоящей статьи, опубликованной в предыдущем номере журнала, основное внимание было уделено созданию гипертекстового транслятора. Кроме того, дано определение гипертекста, которому соответствуют описываемые разработки. При этом детально рассмотрен алгоритм работы транслятора, приведены листинги соответствующих классов, разработан формат базы данных для хранения гипертекста и сопутствующей ему информации. Материал данной части практически полностью основывается на уже изложенном, поэтому ознакомление с первой частью статьи было бы крайне желательно.

Далее будет рассказано об основных принципах работы гипертекстового интерпретатора и о некоторых улучшениях и дополнениях, которые можно внести в приведенные программы.

Как интерпретировать гипертекст

Для того чтобы пользоваться гипертекстовой базой данных, Ваша программа (гипертекстовый интерпретатор) должна уметь ее читать и, прочитав, показывать текст, а также правильно пользоваться перекрестными ссылками. Как уже говорилось ранее, для того чтобы программа могла использовать гипертекст, необходимо только один раз прочесть заголовок базы данных и сохранить его в виде массива (или списка) структур, а именно, объектов класса HTNode (напомню, этот класс уже использовался при построении гипертекстового транслятора, полное описание и листинги были приведены в первой части). Как и в прошлый раз, нас будут интересовать не все поля объекта, а лишь те, которые используются в интерпретаторе (рис. 5, нумерация рисунков и листингов является единой для всей статьи).

KeyWord	
Size	Seek
X	Y
Reference	
PrevTopic	
NextTopic	

Рис. 5. Структура класса HTNode (поля, используемые в гипертекстовом трансляторе)

Объекты HTNode могут содержать информацию как о теме, так и о ссылке. Поля содержат следующую информацию:

`char *KeyWord` – заголовок темы/ссылки;

`long Seek` – адрес во временном файле (смещение от начала файла до начала текста данной темы), только для темы;

int Size	– размер текста темы в строках, только для темы;
HTNode *NextTopic	– указатель на следующую тему в заголовке, только для темы;
HTNode *PrevTopic	– указатель на предыдущую тему в заголовке, только для темы;
HTNode *Reference	– для темы: указатель на первую ссылку из данной темы, – для ссылки: указатель на следующую тему в ссылке;
int X, Y	– координаты первого символа ссылки в тексте относительно 0 строки, 0 позиции, только для ссылки.

(Заметим, что при чтении заголовка поля Reference X и Y остаются нулевыми).

Объекты класса HTHeader (листинг 8) обеспечивают чтение заголовка базы данных, составление массива тем из «кирпичиков» HTNode (конструктор) и поиск в массиве нужной темы (функция Search). Объект HTHeader используется массивом объектов HTNode, так как это гораздо проще, чем более эффективный, с точки зрения быстродействия, метод хранения заголовка в виде сбалансированного двоичного дерева (B-дерева). Объект класса DataFile (листинг 9) может содержать несколько объектов HTHeader, таким образом достигается доступ одновременно к нескольким базам данных. Приведенное описание этого класса весьма неполное: DataFile позволяет лишь добавлять базы данных в набор и пользоваться любой из них, но гипертекстовые ссылки работают только внутри одной базы данных.

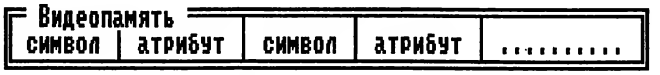
Ядром гипертекстового интерпретатора является объект HyperNode (листинг 10). Конструктор HyperNode, кроме параметров, задаваемых по умолчанию, получает имя файла (FileName), откуда следует читать гипертекст, название темы (NodeName) и указатель на объект DataFile, из которого он сможет получить информацию о заголовке базы данных, соответствующей FileName, откуда, в свою очередь, извлечет информацию о теме NodeName. Эта информация содержится в поле HTNode *ThisHTNode. После чего файл позиционируется по смещению Seek и распаковываются Size строк (функции DecompressText и DecompressLine).

Следующий немаловажный момент — корректная обработка ссылок. Во-первых, при чтении текста необходимо исключить все символы-префиксы. Во-вторых, необходимо каким-то образом запомнить точную позицию в тексте соответствующей ссылки. Есть два принципиально разных пути. Первый достаточно тривиальный: при чтении текста создавать список, содержащий информацию о ссылках. Как Вы уже могли догадаться, мы опять используем для этого класс HTNode с надстройкой из класса HTRef (листинг 11). Указатель на начало списка ссылок поместим в поле ThisHTNode->Reference.

В Вашей прикладной программе при каждом передвижении курсора или «мыши» нужно следить (с помощью функции GetReference), не совпадают ли их новые координаты с координатами X и Y какой-либо из ссылок, а дальше — исключительно дело техники.

Примечательно то, что в поле KeyWord любого элемента из списка ссылок будет находиться часть строки текста данной темы и одновременно заголовок темы, на которую сделана эта ссылка. Подобное представление текста и ссылок не зависит от того, используется в программе текстовый или графический режим.

Если же Ваша программа использует исключительно текстовый режим, можно предложить другой подход к представлению текста и ссылок. Он будет особенно эффективен, если в Вашей гипертекстовой базе данных содержится информация о цветовых атрибутах текста (о них говорилось в разделе «Формат исходного текста»). При чтении текста из базы данных нужно читать не только строки текста, а одновременно с ними и текстовые атрибуты и записывать их в образы видеопамяти, в виде простой последовательности:



При этом ссылка будет выделяться каким-нибудь специальным цветом (атрибутом). Такой формат данных очень удобен для вывода — данные пишутся напрямую в видеопамять. Со ссылками работать тоже довольно просто: необходимо следить за появлением под курсором символов с нужными атрибутами.

В приведенных программах реализованы оба подхода одновременно. В процессе работы функции DecompressText создается массив Text, содержащий собственно текст темы и объект HTRef, хранящий ссылки (что удобно для первого подхода). Но параллельно с этим создается массив Attr, по размерам в точности совпадающий с Text, хранящий информацию об атрибутах текста.

Гипертекстовый интерпретатор должен являться частью программы с хорошим пользовательским интерфейсом, поэтому здесь мы не сможем привести полной работающей программы. Однако можно написать некий прообраз интерпретатора, который будет читать тему из базы данных и составлять список гипертекстовых ссылок. Реализация такого прообраза

интерпретатора гипертекста приведена в листинге 12, функция ProcessTopic.

Что дальше?

Описанные принципы являются некоторым минимумом для того, чтобы создавать гипертексты и работать с ними. Предложенный подход является довольно простым, но вместе с тем написано уже очень много программ, использующих подобные гипертекстовые системы. Можно работать с гипертекстами, используя только описанные принципы, но можно внести и некоторые улучшения, часть из них описана далее.

Улучшение 1. Сжатие данных

В приведенных программах используется алгоритм RLE, который не очень эффективен. Гораздо более эффективным является алгоритм LZW, о чем и упоминают авторы в работе [1]. Вместе с тем алгоритм RLE можно считать неким компромиссом. С одной стороны, он предельно прост в программировании, но дает весьма средние результаты. С другой стороны, в последнее время наметилась широкая тенденция к применению специальных средств по сжатию данных, записываемых на диск в реальном времени (например, STACKER). При этом записывать на диск, обслуживаемый STACKER, данные в нормальном, несжатом виде гораздо выгоднее, чем упаковывать их внешними средствами.

Улучшение 2. Другой тип сжатия данных

Подобная идея использована в гипертекстовых базах данных Tech Help! (by Flambeaux Software, Inc.). Эта идея заключается в следующем: составляется список наиболее часто встречающихся в тексте слов, этот список вписывается в начало базы данных, а затем при каждом появлении этих слов в тексте они замещаются на соответствующую метку замены и номер слова в списке. Это хорошо работает, когда заранее известно, что содержится в тексте и какие слова встречаются в нем наиболее часто.

Улучшение 3. Реорганизация базы данных

Гипертекстовый транслятор, описанный в первой части статьи, обладает одним существенным недостатком: в созданную базу данных нельзя внести никакие изменения. Для того чтобы добавить или удалить какую-нибудь тему или просто откорректировать содержание темы, необходимо иметь исходный текст базы данных.

Улучшение 4. Объединение баз данных

Это в самом деле может понадобиться на практике. Чтобы не объединять исходные тексты, и без того, может быть, очень большие, проще объединить готовые базы данных. При этом необходимо лишь сравнить их заголовки — сделать проверку на совпадение имен тем. Если в объединяемых базах данных нет двух тем с одинаковыми заголовками, нужно соединить заголовки баз данных в один и, соответствующим образом изменив в них адреса тем, скопировать гипертексты в одну базу данных (рис. 6).

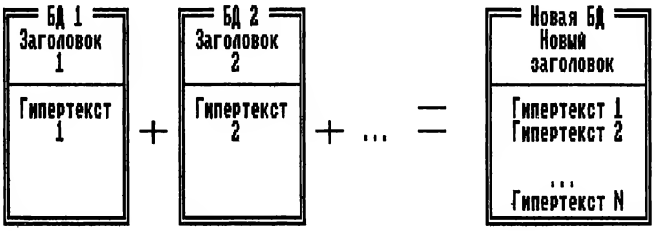


Рис. 6. Объединение гипертекстовых баз данных

Комментарии к листингам

1. Приведенные в статье программные разработки выполнены на языке программирования C++, хотя, естественно, нет совершенно никаких ограничений на использование других языков: авторы работы [1] разработали свою собственную гипертекстовую систему на языке Prolog.

2. Из-за ограничения объема в тексте статьи приведены лишь ключевые моменты программ, основное пояснение алгоритмов перенесено в сами листинги: комментарии составляют до одной третьей всего объема листингов.

3. Константы, общие для всех классов, вынесены в специальный файл HTType.hpp, листинг которого приведен в предыдущем номере журнала.

4. Практически все описанные классы используют класс Error (листинг которого опубликован ранее), обеспечивающий обработку ошибок. В листинге также приведены коды ошибок и тексты, поясняющие сущность той или иной ошибочной ситуации.

Надеюсь, что читатель, прочитав данную статью, найдет для себя ответ на вопросы «а как же оно работает?» и «а как же это делается?». Работая над статьей, автор старался придерживаться двух принципов:

1) дать общее представление того, как программируются системы поддержки гипертекстов, безотносительно к применяемому языку программирования;

2) предоставить программистам-практикам пищу для размышлений в виде готовых прокомментированных листингов на C++, в которых (надеюсь!) захочется разобраться. ■

Литература

1. Д.Гришечкин, В.Оленев, А.Тихонов. Гипертекстовые системы//Монитор.– 1992.– № 5.

Листинг 8а. Файл HTHeader.hpp

```
// Определение класса HTHeader -
// заголовок гипертекстовой базы данных
#ifndef _HTHEADER_
#define _HTHEADER_

#include <HTNode.hpp>
#include <stdio.h>

class HTHeader {
  FILE     *DBase;                          // Файл - база данных
  char*    ReadWord(char *Buf, int MaxLen); // Чтение слова из файла
public:
  HTNode  **NodeList;   // Собственно заголовок - массив
                        // объектов класса HTNode
  char     *FName;      // Имя файла - базы данных
  char     *NetTitle;   // Имя гипертекстовой сети
  unsigned TotalTopics; // Количество тем в заголовке
  int      *ErrCode;    // Код ошибки
           HTHeader(char *fname, int *Err);// Конструктор
           ~HTHeader();                    // Деструктор
  HTNode  *Search(char *keyword);          // Поиск темы в заголовке
};
#endif
```

Листинг 8б. Файл HTHeader.hpp

```
// Класс HTHeader - заголовок гипертекстовой базы данных
#include <HTHeader.hpp>
#include <HTType.hpp>
#include <Error.hpp>
#include <string.h>
#include <ctype.h>  // toupper()

//:::::::::::::::::::::::::::::::::::::::::: Конструктор ::::::::::::::::::::::::::::::::::::::::::
HTHeader::HTHeader(char *fname, int *Err)
{ char buf[80];              // Буфер для различных целей
  ErrCode = Err; NetTitle = NULL;
  TotalTopics = 0; FName = new char[strlen(fname)+1];
                             // Сохраним имя файла
  strcpy(FName,fname);
  if((DBase = fopen(FName,"rb")) = = NULL)          // Откроем базу данных
  { *ErrCode = EvOPENDB; return; }
  if((setvbuf(DBase,NULL,_IOFBF,1024*5))! = 0)  // Установим буферизацию
  { *ErrCode = EvNOEMEM; return; }
  ReadWord(buf,strlen(DBFILE_ID)+1);
                             // Прочтем идентификационную метку
  if(strcmp(buf,DBFILE_ID))  // и проверим ее
  { *ErrCode = EvFILNODB; return; }
  // Прочитаем адрес начала текста тем
  unsigned long DataBeginAddr = getw(DBase)*16+getw(DBase);
  TotalTopics = getw(DBase);    // Прочитаем количество тем
  ReadWord(buf,sizeof(buf));  // Прочитаем заголовок гипертекстовой сети
  if(buf[0]! = '\0')          // Если он есть, сохраним его
  { NetTitle = new char[strlen(buf)+1]; strcpy(NetTitle,buf); }
  // Создадим массив указателей на объекты класса HTNode
  NodeList = new HTNode*[TotalTopics];
  // В цикле прочитаем информацию о темах
  for(unsigned NewTopic = 0; NewTopic < TotalTopics; NewTopic++)
  { ReadWord(buf,fgetc(DBase));    // Заголовок
      int TopicSize = fgetc(DBase); // Размер
      // Смещение от начала файла до начала текста темы
      unsigned long TopicAddr = getw(DBase)*0xFF+getw(DBase)+DataBeginAddr;
      // Создадим новый HTNode
      NodeList[NewTopic] = new HTNode(buf,TopicAddr,TopicSize,NewTopic);
      if(NewTopic > 0)           // Сохраним информацию об окружении
          {                      // Следующий в заголовке
          NodeList[NewTopic-1]->NextTopic = NodeList[NewTopic];
                                 // Предыдущий в заголовке
          NodeList[NewTopic]->PrevTopic = NodeList[NewTopic-1];
          }
  }
  fclose(DBase);
  // Отсортируем заголовок с помощью метода Шелла
  for (unsigned Gap = TotalTopics/2; Gap > 0; Gap/ = 2)
      { for (unsigned i = Gap; i < TotalTopics ; i++)
          { for (int j = i-Gap;
          //Здесь используется переопределенный оператор >
          (*(NodeList[j]) > *(NodeList[j+Gap])) && (j > = 0);j -= Gap)
          { HTNode *tmp = NodeList[j];
          NodeList[j] = NodeList[j+Gap];
          NodeList[j+Gap] = tmp;
          }    // Цикл по j
      }        // Цикл по i
  }    // Цикл по Gap
}

//:::::::::::::::::::::::::::::::::::::::::: Деструктор ::::::::::::::::::::::::::::::::::::::::::
HTHeader::~HTHeader()
{ // По порядку уничтожим все темы в заголовке
  for(unsigned i = 0; i<TotalTopics; i++) delete NodeList[i];
  delete NodeList; delete FName;
  if(NetTitle! = NULL) delete NetTitle;
}

// :::::::::::::::::::::::::::::::::::::::::: Функции - члены ::::::::::::::::::::::::::::::::::::::::::

//.......................................... ReadWord ..........................................
char* HTHeader::ReadWord(char *Buf, int MaxLen)
// Функция читает максимум MaxLen символов в Buf
// Возвращает Buf
{ int i = 0;                             // Счетчик
  do    { Buf[i] = fgetc(DBase);
        } while(Buf[i++]! = '\0' && i<MaxLen);
                                         // Если 0 или достигнут максимум
  Buf[i] = '\0'; return Buf;
}

//.......................................... Search ..........................................
HTNode *HTHeader::Search(char *keyword)
// Функция ищет в заголовке базы данных тему keyword
// методом деления отрезка пополам (дихотомии)
// Возвращает указатель на найденную тему или NULL
{ unsigned LowLimit = 0;                 // Нижняя граница отрезка
  unsigned HighLimit = TotalTopics-1;    // Верхняя граница отрезка
  unsigned Middle;                       // Середина отрезка
  char kwd[80];
  int begin = 0, end = strlen(keyword)-1;
  while(keyword[begin] = = ' ') begin++;       // Удалим начальные пробелы
  while(keyword[end] = = ' ' && begin<end) end--; // и концевые
  for(int i = 0; i<end-begin+1; i++)
        kwd[i] = toupper(keyword[begin+i]);    // Скопируем строку
  kwd[i] = '\0';
  while(LowLimit < HighLimit)              // Пока отрезок не сошелся в точку...
  { Middle = LowLimit+(HighLimit-LowLimit)/2;  // Найдем его середину
        // Используя переопределенные операторы "<" и ">",
        // вычислим, где же находится искомое значение ...
        // ... в "левой" половине
        if(kwd <*(NodeList[Middle])) HighLimit = Middle;
        // ... в "правой" половине
        else if(kwd > *(NodeList[Middle])) LowLimit = Middle+1;
        else return NodeList[Middle]; // А здесь нашлось искомое значение
  }
  return NULL;  // Нужная тема не найдена
}
```

Листинг 9а. Файл DataFile.hpp

```
// Определение класса DataFile -
// поддержка набора гипертекстовых баз данных
#ifndef _DATAFILE_
#define _DATAFILE_

#include <HTHeader.hpp>
#include <stdio.h>

class DataFile {
public:
  int         MaxFiles;     // Максимальное число баз данных
  int         NumFiles;     // Текущее число зарегистрированных баз данных
  int         ErrCode;      // Код ошибки
  HTHeader    **Header;                 // Вектор заголовков баз данных
              DataFile(int amount);     // Конструктор
              ~DataFile();              // Деструктор
  int   AddFile(char *fname);  // Добавление новой базы данных
  int   InList(char *fname);   // Получение номера базы данных в наборе
};
#endif
```

Листинг 9б. Файл DataFile.hpp

```
// Класс DataFile - поддержка набора гипертекстовых баз данных

#include <DataFile.hpp>
#include <HTHeader.hpp>
#include <Error.hpp>
#include <string.h>

//:::::::::::::::::::::::::::::::::::::::::: Конструктор ::::::::::::::::::::::::::::::::::::::::::
DataFile::DataFile(int amount)
// amount - максимально возможное количество баз данных в наборе
{ ErrCode = EvSUCCESS; MaxFiles = amount;
  NumFiles = 0; Header = new HTHeader*[amount];
// Вектор указателей заголовков
}

//:::::::::::::::::::::::::::::::::::::::::: Деструктор ::::::::::::::::::::::::::::::::::::::::::
DataFile::~DataFile()
{ for(int i = 0; i<NumFiles; i++) delete Header[i];
  delete Header;
}
```

Листинг 9б (продолжение)

```
//::::::::::::::::::::::::::::::::::::::::::::: Функции - члены ::::::::::::::::::::::::::::::::::::::::::::
//.....................................................Addfile.....................................................
int DataFile::AddFile(char *fname)
// Функция регистрирует базу данных с именем fname
{ HTHeader *NewHeader;
  // Если уже набрано максимальное количество баз данных
  if(NumFiles > = MaxFiles) { ErrCode = EvTOOMANYDB; return -1; }
  // Если такая база данных уже зарегистрирована
  if(InList(fname) > = 0) { ErrCode = EvDBREG; return -1; }
  // Создадим новый заголовок для этой базы данных
  NewHeader = new HTHeader(fname,&ErrCode); // и зарегистрируем его
  Header[NumFiles] = NewHeader; NumFiles + +;
}
//.....................................................InList.....................................................
int DataFile::InList(char *fname)
// Проверяет, зарегистрирована ли база данных с именем fname
// Возвращает : N    > = 0 -   база данных уже зарегистрирована
//                             под номером N,
//                   <    0 -  не зарегистрирована
{ strupr(fname);
  for(int i = 0; i < NumFiles; i + +)
      if(!strcmp(fname,Header[i] -> FName)) return i;
  return -1;
}
```

Листинг 10а. Файл Hyper.hpp

```
// Определение класса HyperNode - полная информация
// об узле гипертекстовой сети
#ifndef _HYPERNODE_
#define _HYPERNODE_

#include <DataFile.hpp>
#include <HTNode.hpp>
#include <HTRef.hpp>
#include <Error.hpp>

// Цвета текста и ссылок, устанавливаемые по умолчанию (a la Borland)
#define DefaultAttributes "\x30\x31\x1B\x2F\x9B"
#define DefaultRefAttrib 0x1E

class HyperNode {
  int   DecompressText(); // Создание текста темы
  // Создание (распаковка) одной строки текста
  int       DecompressLine(int i, char *text, char *attr);
  char      Attributes[5]; // Определяемые пользователем атрибуты текста
                           // (фонты, цвета и так далее)
  char      RefAttrib;     // Определяемый пользователем атрибут ссылок
  FILE      *DBFile;       // База данных
public:
  HTRef     *References;   // Поддержка ссылок
  HTNode    *ThisHTNode;   // Краткая информация об узле
                           // берется из заголовка
  char      **Text;        // Текст темы
  char      **Attr;        // Атрибуты темы
  int       RefCount;      // Количество ссылок в теме
  short     Completed;     // Флаг успешного завершения
  int       *ErrCode;      // Код ошибки
                           // Конструктор
            HyperNode(DataFile *, char *filename, char *nodename,
                      char* = DefaultAttributes, char = DefaultRefAttrib);
            ~HyperNode(); // Деструктор
  char*     GetReference(int x, int y);    // Проверяет наличие ссылки в (x,y)
  char*     GetNextHyperNode();            // Следующая ссылка в базе данных
  char*     GetPrevHyperNode();            // Предыдущая ссылка
};
#endif
```

Листинг 10б. Файл Hyper.cpp

```
// Класс HyperNode - полная информация об узле гипертекстовой сети

#define TEXTMAXLEN 255     // Максимальная длина строки текста

#include <Hyper.hpp>
#include <HTNode.hpp>
#include <HTType.hpp>
#include <string.h>
#include <ctype.h>

//::::::::::::::::::::::::::::::::::::::::::::: Конструктор ::::::::::::::::::::::::::::::::::::::::::::
HyperNode::HyperNode(DataFile *DF, char *fname, char *nodename,
                     char *attribs, char refattr)
// Создает текст и ссылки темы nodename из файла fname,
// зарегистрированного в DataFile *DF
// По окончании Completed = = 0 в случае ошибки
{ Completed = 0; ErrCode = &DF-> ErrCode;
  ThisHTNode = NULL; strcpy(Attributes,attribs);   // Сохраним атрибуты
  RefAttrib = refattr; strupr(nodename);
  int FileIndex;        // Проверим, зарегистрирован ли файл
  if((FileIndex = DF-> InList(fname)) < 0)
  {    *ErrCode = EvDBNOTREG; return; }
  // Найдем нужную ссылку в заголовке
  if((ThisHTNode = DF-> Header[FileIndex]-> Search(nodename)) = = NULL)
  {    *ErrCode = EvTOPNOFD; return; }
  References = new HTRef(ThisHTNode); // Создадим обработчик ссылок
  // Откроем файл
  if((DBFile = fopen(DF-> Header[FileIndex]-> FName,"rb")) = = NULL)
  {    *ErrCode = EvOPENDB; return; }
  if(DecompressText() = = 0)                // Распакуем текст
  {    *ErrCode = EvREADCOMP; return; }
  fclose(DBFile);
  RefCount = References-> RefCount;     // Для удобства скопируем это значение
  Completed = 1;                        // Ошибок не обнаружено
}

//::::::::::::::::::::::::::::::::::::::::::::: Деструктор ::::::::::::::::::::::::::::::::::::::::::::
HyperNode::~HyperNode()
{ delete References;          // Уничтожим обработчик ссылок и сами ссылки
  if(Text)                    // Если текст был создан
  {    for(int i = 0;i < = ThisHTNode-> Size;i + +)
       {                              // Мы уничтожим каждую строку
       if(Text[i]) delete Text[i];    // текста и
           if(Attr[i]) delete Attr[i]; // атрибутов
       }
       if(Text) delete Text; if(Attr) delete Attr;
  }
}

//::::::::::::::::::::::::::::::::::::::::::::: Функции - члены ::::::::::::::::::::::::::::::::::::::::::::
//..................................... DecompressText .....................................
int HyperNode::DecompressText()
// Функция распаковывает текст и атрибуты из базы данных
// Возвращает:       > 0 в случае успеха,
//                   0 при ошибке
{ char *ret;                          // Используется для выявления ошибок
  char buf_text[TEXTMAXLEN],    // Буфер для текста
       buf_attr[TEXTMAXLEN];    // Буфер для атрибутов
  Text = new char*[ThisHTNode-> Size + 1];  // Создадим массивы для текста
  Attr = new char*[ThisHTNode-> Size + 1];  // и атрибутов
  fseek(DBFile,ThisHTNode-> Seek,SEEK_SET);  // Позиционируем файл
  if(fgetc(DBFile)! = SYNCH_BYTE)       // Проверим байт синхронизации
  {    *ErrCode = EvTBEGNOFD; return 0; }
  for(int i = 0; i < ThisHTNode-> Size; i + +)          // В цикле распакуем текст
  { ret = fgets(buf_text,sizeof(buf_text),DBFile);      // Прочтем строку
       if(ret = = NULL) break;                          // Проверим ошибку
       DecompressLine(i, buf_text, buf_attr);           // Распакуем
       Text[i] = new char[strlen(buf_text) + 1];        // Скопируем текст
       strcpy(Text[i],buf_text);
       Attr[i] = new char[strlen(buf_attr) + 1];        // и атрибуты
       strcpy(Attr[i],buf_attr);
  }
  Text[ThisHTNode-> Size + 1] = NULL;                   // Последнюю строку текста
                                                        // и атрибутов
  Attr[ThisHTNode-> Size + 1] = NULL;   // обнулим - так будет легче работать
  return (ret! = NULL);                 // Возвратим код ошибки
}

//.....................................DecompressLine.....................................
int HyperNode::DecompressLine(int Nline, char *s, char *a)
// Функция распаковывает строку гипертекста, управляя
// атрибутами и ссылками
// Параметры : Nline -    номер распаковываемой строки от начала темы,
//             s -        строка текста,
//             a -        строка атрибутов.
// Возвращает длину распакованной строки
{ char text[TEXTMAXLEN], attr[TEXTMAXLEN]; // Буферы
  int NeedFillRef = 0;              // Флаг = = 1, когда обрабатывается ссылка
  int NeedAddRef = 0;               // Флаг = = 1, только в начале ссылки
  char CurrentAttr = Attributes[0];         // Текущий атрибут
  for(int i = 0, j = 0; i < strlen(s); i + +)   // В цикле просмотрим строку
  {    if(s[i] = = COMPRESS)                    // Метка сжатия
       {                                        // Распакуем текст
       for(char Count = 0; Count < (char)(s[i + 1]-DELAY); Count + +)
            {    text[j + Count] = s[i + 2]; attr[j + Count] = CurrentAttr;
            }
       j + = Count; i + = 2;                    // Увеличим счетчики
       } else if(s[i] = = TYPEREF)              // Метка ссылки
       {    if(s[i + 1] = = TYPEREF)            // Ситуация "&&"
            {    text[j] = s[i]; attr[j] = CurrentAttr; }
            else if(NeedFillRef)                // Конец ссылки
                 {    CurrentAttr = Attributes[0];  // Сбросим атрибут
                 References-> FillReference(j-NeedFillRef); // Запомним ссылку
                 NeedFillRef = 0;               // Сбросим флаг
                 } else                         // Начало ссылки
                 {    CurrentAttr = RefAttrib; // Установим атрибут
                      NeedAddRef = 1;          // Установим флаг
                 }
            }    // Конец обработки ссылок
            else if(s[i] = = TYPECHAR)          // Метка спецсимвола
```

```
            {   text[j] = s[i+1]; attr[j] = CurrentAttr; j++; i++; }
            else if(s[i] = = TYPESPECIAL || s[i] = = TYPEATTRIB)
                                                // Метка атрибута
            {   if(s[i+1] = = TYPESPECIAL && s[i] = = TYPESPECIAL)
                                                // Ситуация "^^"
                {   text[j] = s[i]; attr[j] = CurrentAttr; continue; }
                if(s[i] = = TYPESPECIAL) switch(toupper(s[i+1]))
                { // Предопределенные атрибуты
                    case TYPEC1: CurrentAttr = Attributes[0]; break;
                    case TYPEC2: CurrentAttr = Attributes[1]; break;
                    case TYPEC3: CurrentAttr = Attributes[2]; break;
                    case TYPEC4: CurrentAttr = Attributes[3]; break;
                    case TYPEC5: CurrentAttr = Attributes[4]; break;
                } else if(s[i] = = TYPEATTRIB) CurrentAttr = s[i+1];
                                                // Просто атрибут
                i++;
            } // Конец обработки атрибутов
            else if(isprint(s[i]))              // Обычные символы
            {   if(NeedAddRef = = 1)    // Надо добавить ссылку
                {   NeedFillRef = j;    // Установим флаги
                    NeedAddRef = 0;     // и добавим ссылку
                    References->AddReference(Nline,j,&text[j]);
                }
            text[j] = s[i]; attr[j] = CurrentAttr; j++;     // Просто символ
            }   // Конец обработки обычного символа
    }   // Конец обработки строки
    // Ссылка осталась несохраненной - сохраним ее
    if(NeedFillRef) References->FillReference(j-NeedFillRef);
    // По окончании строки не сброшен атрибут или строка пустая
    if(CurrentAttr! = attr[j-1] || j = = 0)
    {   attr[j] = CurrentAttr;      text[j] = ' ';      j++; }
    text[j] = '\0';         // Установим завершающие нули
    attr[j] = '\0';
    strcpy(s,text);         // Скопируем строки
    strcpy(a,attr);
    return(j);              // Возвратим значение
}
//.....GetReference.....
char* HyperNode::GetReference(int x, int y)
// Проверяет, нет ли ссылки в позиции (x,y)
// Возвращает NULL, если там нет ссылки, саму ссылку, если она там есть
{ HTNode *a = ThisHTNode->Reference;
  while(a! = NULL)                  // Пройдем по списку ссылок
  {   if(a->Y = = y)                // Если строка та же
      {                             // Если и позиция в строке подходит
      if(x> = a->X && x< = (a->X+strlen(a->KeyWord)))
          return a->KeyWord;        // Ссылка найдена
      }
      a = a->Reference;
  }
  return NULL;                      // Ссылка не найдена
}
//..... GetNextHyperNode .....
char* HyperNode::GetNextHyperNode()
// Возвращает следующую ссылку в базе данных
// или NULL, если данная ссылка последняя
{ return ThisHTNode->NextTopic->KeyWord; }
//..... GetPrevHyperNode .....
char* HyperNode::GetPrevHyperNode()
// Возвращает предыдущую ссылку в базе данных
// или NULL, если данная ссылка первая
{ return ThisHTNode->PrevTopic->KeyWord; }
```

Листинг 11а. Файл HTRef.hpp

```
// Определение класса HTRef - поддержка ссылок гипертекстовой сети
#ifndef _HTREF_
#define _HTREF_
#include <HTNode.hpp>
class HTRef {
  char *LastRefName;      // Собственно ссылка (часть текстовой строки)
  HTNode  *ParentHTNode;  // Тема, откуда сделана ссылка
  HTNode  *LastRef;       // Последняя ссылка в списке
public:
  int   RefCount;                   // Количество ссылок
        HTRef(HTNode *parent);      // Конструктор
        ~HTRef();                   // Деструктор
// Добавление ссылки в список
  void AddReference(int Nline, int Npos, char *s);
  void FillReference(int refsize);  // Заполнение поля KeyWord ссылки
};
```

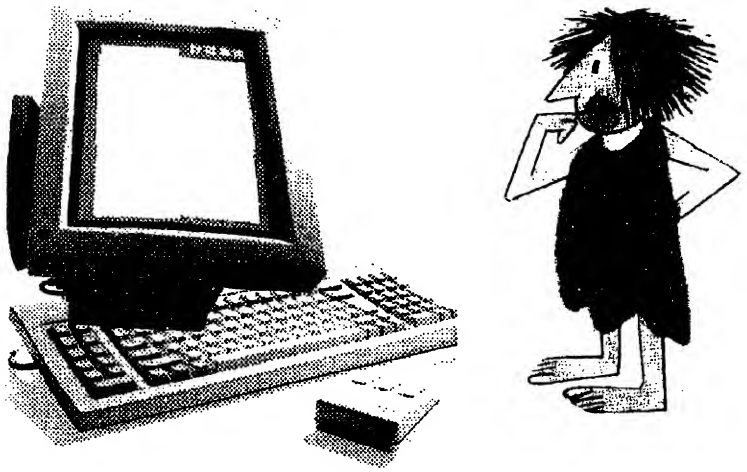
Листинг 11б. Файл HTRef.cpp

```
// Класс HTRef - поддержка ссылок гипертекстовой сети
#include <HTRef.hpp>
#include <HTNode.hpp>
#include <string.h>
//::::::::::::::::::::::::::::::::::::::::::::::: Конструктор ::::::::::::::::::::::::::::::::::::::::::::::
HTRef::HTRef(HTNode *parent)
// parent - тема, откуда сделана ссылка
{ ParentHTNode = parent; LastRef = NULL;
  LastRefName = NULL;   RefCount = 0;
}
//::::::::::::::::::::::::::::::::::::::::::::::: Деструктор ::::::::::::::::::::::::::::::::::::::::::::::
HTRef::~HTRef()
{ LastRef = ParentHTNode->Reference;
  while(LastRef! = NULL)        // Уничтожим список ссылок
  {   HTNode *TmpDel = LastRef;
      LastRef = LastRef->Reference; delete TmpDel;
  }
  // Обнулим поле ссылки в родительской теме
  ParentHTNode->Reference = NULL;
}
//::::::::::::::::::::::::::::::::::::::::::::::: Функции - члены ::::::::::::::::::::::::::::::::::::::::::::::
//..... AddReference .....
void HTRef::AddReference(int Nline, int Npos, char *s)
// Функция добавляет ссылку в список
{ HTNode *NewNode = new HTNode(NULL,0,0,0,Npos,Nline);
  if(LastRef = = NULL) ParentHTNode->Reference = NewNode;
  else LastRef->Reference = NewNode;
  LastRef = NewNode; LastRefName = s;
}
//..... FillReference .....
void HTRef::FillReference(int refsize)
// Функция завершает процесс создания новой ссылки,
// заполняя поле KeyWord
// refsize - длина слова
{ LastRef->KeyWord = new char[refsize];
  strncpy(LastRef->KeyWord,LastRefName,refsize);
  LastRef->KeyWord[refsize] = 0;   strupr(LastRef->KeyWord);
}
```

Листинг 12. Файл HTView.cpp

```
// Прообраз гипертекстового интерпретатора
#include <DataFile.hpp>
#include <Hyper.hpp>
#include <string.h>
#include <stdio.h>
// Обработка одной темы
int ProcessTopic(DataFile *HBase, char *FName, char *TopicName)
{ HyperNode *Topic = new HyperNode(HBase,FName,TopicName);
  if(HBase->ErrCode! = 0) return 0;// Были ошибки
  for(int j = 0; j < Topic->ThisHTNode->Size; j++)      // Выведем текст
        printf("%s\n",Topic->Text[j]);
  // Выведем ссылки и их координаты
  HTNode *ref = Topic->ThisHTNode->Reference;
  while(ref! = NULL)
  {   printf("Ссылка \"%s\" %d %d\n",ref->KeyWord,ref->X,ref->Y);
      ref = ref->Reference;
  }
  delete Topic;
  return 1;
}
int main(int argc, char *argv[])
{ printf("Прообраз гипертекстового интерпретатора\n");
  if(argc! = 2)
  {   printf("Использование: HTVIEW DataBase\n"); return 1; }
  DataFile *HyperBase = new DataFile(5);
  HyperBase->AddFile(argv[1]);
  printf("Файл: %s\n",argv[1]);
  if(HyperBase->Header[0]->NetTitle)
        printf("Название сети: %s\n",HyperBase->Header[0]->NetTitle);
  printf("%d тем\n",HyperBase->Header[0]->TotalTopics);
  for(int i = 0; i < HyperBase->Header[0]->TotalTopics; i++)
        printf("Тема N %d: \"%s\"\n", i,
HyperBase->Header[0]->NodeList[i]->KeyWord);
  char Name[80];
  printf("Введите имя темы:"); scanf("%s",Name);
  int result = ProcessTopic(HyperBase,argv[1],Name);
  if(result = = 0) printf("Ошибка N %d\n",HyperBase->ErrCode);
  delete HyperBase;
  return 0;
}
```

Работа над ошибками



Д. Арапов

Введение

При выполнении программ могут возникать исключительные (их еще называют «нештатные») ситуации. Причины их появления разнообразны. Ими могут быть: сбой внешнего устройства, попытка деления на 0, нехватка какого-либо ресурса, например, места на диске при попытке записи файла или неинтегрируемая особенность у подинтегральной функции. Источником исключительных ситуаций может быть пользователь, задавший неверные входные данные, или программная ошибка. Программы должны распознавать нештатные ситуации и предусматривать реакцию на них. Цель данной статьи — систематизировать опыт обработки нештатных ситуаций в программах на Turbo Pascal.

От романтики к реализму

Разработка любой серьезной программы включает несколько стадий. Вначале, когда пишется прототип, внимание уделяется основным алгоритмам и пользовательскому интерфейсу. Однако по мере перехода от программы-прототипа, программы-идеи к конечному продукту главной заботой разработчика становится отслеживание всевозможных исключительных ситуаций, способных помешать нормальному выполнению программы. Пользователь вправе требовать от нее «правильного» поведения независимо от обстоятельств. Нельзя рассчитывать на то, что программа будет работать в любых условиях. У всякой программы есть пределы применимости. Однако отчуждаемая программа должна удовлетворять, по крайней мере, двум требованиям: во-первых, от нее требуется сохранять постоянные данные в порядке (особенно это касается тех из них, которые не могут быть восстановлены из других программным путем), а во-вторых, в случае прекращения работы, она обязана внятно сообщить пользователю, что произошло.

Поясним эти требования на примере программы копирования файлов. Применительно к ней разумно требовать, во-первых, чтобы она не портила исходный файл, даже если по ошибке одно и то же имя было задано и в качестве параметра-источника, и в качестве параметра-результата, а во-вторых, при нехватке места на диске она должна сообщить пользователю, что файл не может быть записан, так как исчерпано дисковое пространство.

Частным (но очень важным) примером первого требования будет корректная работа с динамически распределяемой памятью. Если подпрограмма захватила участок памяти, то в случае аварийного завершения она должна вернуть его назад в пул свободной памяти. Не делать этого можно только в одном случае, когда аварийно завершается вся программа. В этом случае можно не заботиться о целостности как локальной (принадлежащей программе), так и глобальной динамически распределяемой области памяти (heap). Локальная прекратит свое существование вместе с программой, а о возврате блоков памяти в глобальный пул позаботится операционная система. Во всех других случаях рекомендуется крайне внимательно отнестись к этой проблеме, поскольку неполное освобождение памяти в нештатных ситуациях, как правило, проявляется не сразу. Эта ошибка не заметна при отладке, а для того, чтобы предусмотреть тест, в котором эта нештатная ситуация повторяется в цикле достаточное для исчерпания памяти число раз, надо быть виртуозом тестирования.

Характер обработки исключительных ситуаций и ее роль в структуре программы существенно зависят от класса программы. Прекращение работы с выдачей диагностики — вполне разумный вариант реакции на ошибку для небольших программ. Однако это решение совершенно неприемлемо для большинства интерактивных программ и систем реального времени. Представьте себе, скажем, такой кошмарный сценарий: программа управления ядерным реактором неожиданно прекращает работу с выдачей сообщения «Ошибка 20561: переполнение кольцевого буфера внешних событий». Можно привести и более прозаический пример: работая с редактором, Вы пытаетесь сохранить текст. Вдруг окно подпрыгивает на несколько строк вверх, и Вы оказываетесь в среде DOS, где с удивлением читаете: «Error: disk full». Набранный вами текст пропал.

Из чего состоит работа над ошибками?

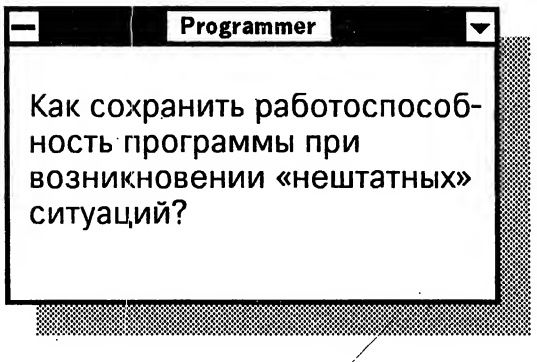
Перед создателями программ перечисленных типов встает вопрос: «Как сохранить работоспособность программы при возникновении исключительной ситуации?»

Итак, посмотрим, что должен предусмотреть автор программы на случай непредвиденных обстоятельств. Как правило, следует организовать деятельность, состоящую из трех фаз.

1. Фаза распознавания нештатной ситуации и возбуждения исключения.

2. Фаза распространения исключения, то есть передачи управления от точки распознавания ситуации к точке ее обработки.

3. Фаза обработки исключительной ситуации. Именно эта фаза отвечает за восстановление при ошибке и выдачу диагностики.



Наибольший интерес вызывает случай, когда точка возбуждения и точка обработки разнесены: нештатная ситуация распознается в вызванной программе, а обработана может быть только в вызывающей. Более того, процедуру, способную обработать исключение, и ту, в которой оно возбуждено, достаточно часто разделяет несколько уровней в стеке вызовов. Отметим несколько проблем, возникающих в этом случае.

1. Одно и то же с точки зрения вызываемой подпрограммы событие может рассматриваться вызывающей процедурой как исключение, а может и не рассматриваться. Иными словами, исключение для одних может оказаться правилом для других.

2. Если исключение вызывается в подпрограмме большого уровня вложенности, то в вызывающей программе его может быть трудно интерпретировать. Например, исключение может быть описано в терминах другого модуля и зависеть от его реализации.

3. Исключительная ситуация может нарушить целостность данных, имеющих больший срок жизни, чем процедура, в которой обрабатывается исключение. Обработка исключений должна предусматривать восстановление целостности таких данных. Наиболее неприятен тот случай, когда нарушение целостности затрагивает несколько программных модулей и восстановление нельзя провести в одном месте.

Лучшее лекарство — профилактика

Источником исключительных ситуаций могут быть как ошибки в программе, так и неверные входные значения, заданные пользователем. Можно считать, что тщательная отладка и тестирование способны устранить ошибки первой группы, ошибки же во входных данных исключить никогда нельзя. В полностью отлаженной программе можно подавить проверки, которые делаются для обнаружения ошибок первой группы. Вместо удаления строк исходного текста, содержащих проверки, удобнее воспользоваться механизмом условной компиляции, что позволит при необходимости без труда вновь включить контроль. Обычная реакция на ошибку этой группы состоит в выдаче диагностического сообщения.

Обратимся к ошибкам во входных данных. Часть таких ошибок можно легко отсечь с помощью контроля входных данных. Такой контроль обычно организуют непосредственно вслед за вводом данных, а когда речь идет не о программе в целом, а об отдельном модуле, то контролируют входные параметры процедур. Входной контроль способствует раннему обнаружению исключительных ситуаций и позволяет предотвращать нарушение целостности данных. Во всех случаях, когда он возможен, им не следует пренебрегать.

В некоторых случаях входной контроль требует почти столько же времени, сколько и само действие. Типичный пример — процедура интегрирования. Выяснение интегрируемости функции-параметра может потребовать столько же времени, сколько и само интегрирование. В таких случаях операции контроля включают непосредственно в процесс счета, и именно в этих случаях и возникает большинство проблем, связанных с обработкой нештатных ситуаций.

Критерии оценки

Прежде чем обсуждать конкретные приемы работы над ошибками, перечислим требования, предъявляемые к механизму обработки исключений.

1. **Простота.** Механизм не должен становиться самостоятельным источником ошибок.

2. **Универсальность.** Механизм должен быть применим к нештатным ситуациям разной природы, возникающим в разных программных единицах: процедурах, функциях, конструкторах, деструкторах, телах модулей.

3. **Удобочитаемость.** Механизм должен позволять читателю программы легко отличать основное русло алгоритма от кода обработки нештатных ситуаций.

4. **Эффективность.** Наличие механизма не должно замедлять нормального (штатного) выполнения программы.

5. **Гибкость.** Необходимо иметь возможность интерпретировать одно и то же событие по-разному: и как аварию, и как норму.

6. **Прозрачность.** Если процедура А вызывает процедуру В, которая в свою очередь вызывает процедуру С, где может быть возбуждена исключительная ситуация, обрабатываемая в А, то механизм должен позволять не включать в В специального кода на этот случай.

Turbo Pascal не содержит языковых средств для работы с исключениями. Программисты используют для этих целей различные механизмы. Перечислим их и попытаемся сравнить с точки зрения перечисленных требований. Чтобы обосновать оценки, поставленные методам, к описанию методов прилагаются примеры фрагментов программ. Оценки приводятся по 4-балльной шкале:

1 — неудовлетворительно;
2 — приемлемо;
3 — хорошо;
4 — очень хорошо.

1. Останов программы с помощью Halt

Это самый простой способ обработки ошибок. На экран выдается диагностическое сообщение, после чего вызывается встроенная процедура Halt. Чтобы оповестить DOS об исключительной ситуации, рекомендуется установить параметр этой процедуры в ненулевое значение. Возможно предусмотреть и некоторые «предсмертные» действия. Для этого их необходимо оформить в виде процедуры (обязательно FAR!) и установить на нее указатель ExitProc (unit System). Ваша процедура будет вызвана при аварийном завершении. Это может быть полезно, например, когда программа использует собственные буферы файлов. В процедуре выхода можно предусмотреть принудительный сброс такого буфера. При необходимости можно задать несколько процедур выхода и связать их в цепочку.

Пример 1

Проблема: при аварийном завершении (независимо от его причины) программа должна сбросить буфер в файл.
В теле модуля, отвечающего за работу с этим файлом, необходимо описать процедуру-обработчик. В данном случае это MyExitProc. Указатель на старый обработчик сохраняется в переменной OldExitProc.

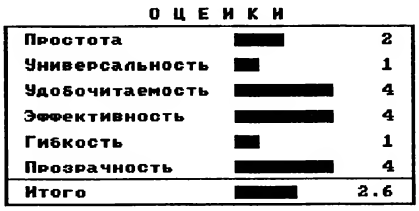
```
var OldExitProc : pointer;
{ описание файла, буфера etc.}

procedure MyExitProc; far;
begin { MyExitProc }
    Write(TheFile,Buffer);          {выполняем сброс буфера}
    ExitProc = OldExitProc;         {восстанавливаем старую
                                     процедуру выхода}
end; { MyExitProc }

begin { тело модуля работы с файлом }
    OldExitProc := ExitProc;        {запоминаем старую
                                     процедуру выхода}
    ExitProc := @MyExitProc;        {устанавливаем новый
                                     обработчик}
    {инициализируем модуль}
end.
```

В процедуре или функции, где может возникнуть нештатная ситуация, появляется следующий код:

```
if ErrorCondition then begin
    WriteLn('Ошибка: сломалось то-то и то-то.');
    Halt(1); {Общепринятых соглашений по кодировке ошибок
              в DOS нет}
end; { if }
```



2. Возвращение специально оговоренного значения

Механизм состоит в том, что одно (реже несколько) «невозможное» значение резервируется для указания на ошибку. Оно возвращается функцией в качестве результата или процедурой в виде var-параметра. Чаще всего это -1, 0 или NIL. В вызывающей процедуре предусматривается ветвление в соответствии с полученным значением.

Пример 2

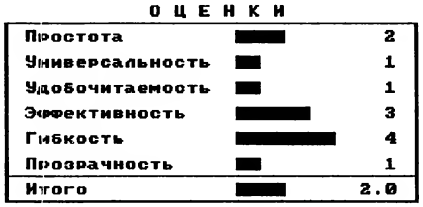
Проблема: внутри программы вызывается функция поиска, возвращающая индекс искомого элемента в массиве. Вызывающая программа рассматривает его отсутствие как ошибку (функция поиска «не знает», что это так). Как и в предыдущем примере, при аварийном завершении требуется сбросить буфер в файл.

```
function Search( Value: integer ): integer;
var index : integer;
begin { Search }
    index := 0;
    while (index < Count) and (Items[index] <> Value) do Inc(index);
    if index < Count
        then Search := Index
    else Search := -1; {оговоренное нелегальное значение}
end; { Search }
```

Если вызывающая программа в состоянии восстановиться после ошибки, то следует кодировать примерно так:

```
Position := Search;
if Position = -1
    then HandleError( erSearchError )
else begin
end; { if }
```

Чтобы код для ошибочных ситуаций не загромождал программу, обработку ошибочных ситуаций вынесли в специальную процедуру HandleError.



Наиболее привлекательным свойством этого подхода является гибкость. Результат может интерпретироваться как ошибка в соответствии с пожеланиями вызывающей процедуры. Самый существенный недостаток — отсутствие универсальности. У процедур отсутствует возвращаемое значение. Кроме того, для функций во многих случаях нельзя выделить ни одного «невозможного» значения. Ввиду этого систематически применять этот подход нельзя, однако в его защиту следует сказать, что им часто пользуются.

3. Преобразование процедуры в функцию, возвращающую код завершения

Хотя этот способ не требует «невозможного» значения, он применим только к процедурам. Опция компиляции {$X-}, позволяющая использовать синтаксис вызова процедур для вызова функций, дает возможность анализировать результат функции в вызывающей программе только в случае необходимости.

Это простой и гибкий механизм. Как и предыдущий, он довольно эффективен, особенно в тех случаях, когда не требуется распространения исключения далеко вверх по стеку вызовов. По этой схеме устроена обработка ошибок в библиотеке Paradox Engine для Pascal.

Пример 3

Проблема: процедура обновления базы данных может закончиться неудачно по многим причинам: обнаружен дублированный ключ, исчерпано место на диске и так далее. Аварийные действия могут быть различны для разных ситуаций. Согласно изложенному выше, процедура преобразована в функцию, возвращающую код ответа.
Вызывающая программа обычно имеет вид:

```
case Update( Rec, Position ) of
    0:begin
    {Действия при нормальном завершении обновления базы}
    end;
erKeyDuplicated:
    begin
    {Реакция на эту ошибку локализована в этой процедуре}
    end;
    else
    begin
    {Частичная обработка ошибки}
        exit    {Распространение исключения.
                 Код исключения утерян.}
    end;
end; { case }
```

Если код исключения нужно сохранить, скажем, для вызывающей программы, то вместо строки

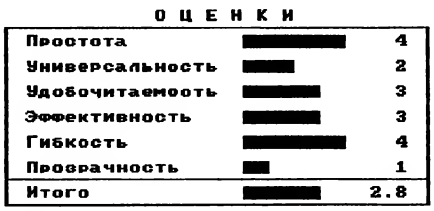
```
case Update( Rec, Position ) of
```

следует использовать две:

```
RC := Update( Rec, Position );
case RC of
```

Когда есть уверенность, что исключения не может возникнуть (увы! в жизненных примерах, подобных приведенному, это бывает довольно редко), можно просто вызвать

```
Update( Rec, Position );
```



4. Код возврата в качестве отдельного var-параметра

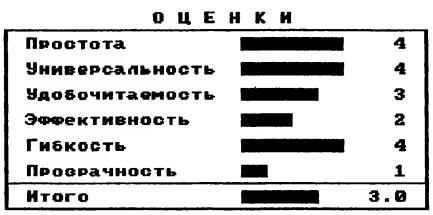
Вместо того чтобы передавать код возврата в виде результата функции, его можно передать в качестве var-параметра. Этот способ универсальнее предыдущих и применим как к процедурам, так и к функциям, конструкторам и деструкторам. Вместе с тем этот способ менее эффективен, чем предыдущий, так как требует передачи адреса дополнительного параметра.

Пример 4

Пример 3 можно закодировать с передачей кода возврата в качестве отдельного var-параметра. При этом «потерять» его (как в предыдущем случае) нельзя. Фрагмент вызывающей программы будет иметь вид:

```
Update( Rec, Position, RC );
case RC of
```

Далее как в предыдущем случае. Если фактический и формальный параметры имеют одно и то же имя, то для транзитной передачи кода возврата наверх, в вызывающую программу, не требуется предпринимать специальных действий.



5. Код возврата в виде переменной модуля

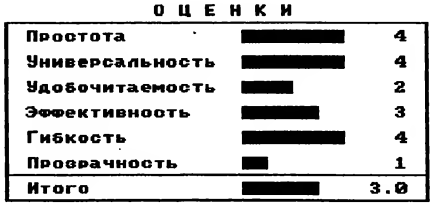
В одном из модулей можно выделить специальную переменную, в которую при нештатной ситуации заносится ее код. Этот способ весьма эффективен и часто используется. Предыдущим способам он уступает лишь по удобочитаемости.

Пример 5

Рассматриваемый пример легко закодировать, используя код возврата в виде переменной модуля:

```
Update( Rec, Position );
case RC of {ветвление по коду возврата}
```

Удобочитаемость несколько страдает, требуются дополнительные усилия, чтобы понять, кто изменяет переменную RC.



Как видно из таблиц, перечисленные выше механизмы имеют общий недостаток. Они не удовлетворяют требованию прозрачности. Если глубина стека велика, а исключение, возбуждаемое в одной из процедур нижнего уровня, может быть обработано только в одной из процедур верхнего уровня, то для распространения исключения вверх по стеку вызовов от точки возбуждения до точки обработки на каждом из уровней потребуется ветвление. Иными словами, вызов процедуры или функции, в недрах которой может

случиться авария, должен сопровождаться конструкцией

```
if Ошибка then exit;
```

Если рассматривать достаточно сложные программы, то окажется, что большинство вызовов требует такого обрамления. Это делает программу более длинной, менее понятной и эффективной. Если оставаться в рамках языка Pascal, то взамен ничего предложить нельзя, однако если воспользоваться встроенным ассемблером и некоторыми элементарными знаниями о структуре стека вызовов, то можно организовать непосредственный переход из процедуры, где произошло исключение, в процедуру, где имеется обработчик, минуя промежуточные уровни стека вызова. Пример подобных действий (процедуры SetJump и LongJump) имеется в пакете Turbo Professional фирмы Turbo Power. Воспользовавшись заложенными в нем идеями, можно реализовать механизм работы с исключениями, который близок к соответствующему механизму языка Ada.

6. Использование механизма исключений

Механизм реализуется с помощью модуля Events, который приводится в листинге 1. Исключительным ситуациям, которые различаются в программе, сопоставляются константы длиной в слово (кроме 0). Возбуждение исключения имеет вид

```
Events.Raise ( Exc : word );
```

В процедуре верхнего уровня, в которой вызываются процедуры более низкого уровня и производится обработка исключений, нужно использовать следующую конструкцию:

```
case Events.ExcBegin of
    exOk: begin
    {* Действия, которые могут создать нештатные ситуации *}
        Events.ExcEnd;
    end;
    exE1: begin
        {* Обработчик исключения exE1 *}
    end;
    else begin
        {* Обработка для всех других исключений. *}
    end;
end; { case }
```

Поскольку основные законы языка Pascal нарушены, следует объяснить порядок выполнения этой конструкции.

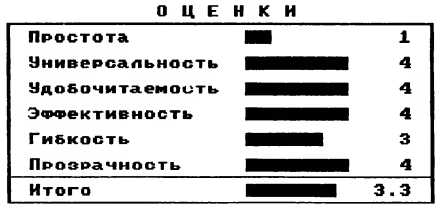
Вычисляется функция ExcBegin. Она запоминает текущие значения регистров BP, SP, IP и CS. При входе в область обработки исключений она всегда возвращает значение exOk (= 0). Далее выполняется ветвь exOk. Она описывает нормальную последовательность действий в предположении, что никаких исключительных ситуаций нет. Она может содержать вызовы функций и процедур (в том числе и рекурсивные). Если в процессе выполнения этой ветви не возникнет аварийных ситуаций (не будет обращений к процедуре Raise), то будет выполнена служебная процедура ExcEnd и управление перейдет на конец оператора case. Последовательность выполнения будет совершенно необычной, если внутри этой ветви (безразлично, на каком уровне вложенности) возникнет нештатная ситуация и будет выполнена процедура Raise. В этом случае управление вернется *внутрь* функции ExcBegin, но на этот раз она возвратит ненулевой код исключительной ситуации (тот самый, который был передан процедуре Raise в качестве параметра). Далее все пойдет в соответствии с правилами языка. Управление будет передано той ветви оператора case, которая помечена соответствующей константой, а в случае ее отсутствия — ветви else.

Замечание. Допускаются вложенные области обработки исключений.

Практический пример использования модуля Excs приведен в листинге 2.

Наиболее существенным недостатком предложенного подхода является его сложность. С ней связана возможность ошибок в силу следующих причин.

1. Переполнение внутреннего стека областей обработки исключений, его размер задается константой и не может быть изменен во время выполнения. (Альтернативная реализация с использованием динамической памяти также не лишена этого недостатка.)
2. Пропущенные обращения к ExcEnd. Эта процедура — чисто служебная, она обеспечивает корректность работы со стеком исключений.
3. Невозможность работы с оверлеями.
4. Неприменимость в конструкторах объектов.



Кроме этого, данный подход несколько многословен, что ограничивает его применимость в простых случаях. Однако у этого механизма имеются и существенные преимущества: в случае написания программ с большой глубиной стека вызовов он позволяет создавать эффективные и читаемые программы.

Обработка ошибок и объектно-ориентированное программирование

Объектно-ориентированный подход помог решить многие проблемы технологии программирования, но не в области обработки нештатных ситуаций. Использование объектов, являющихся, как правило, динамическими структурами данных с большим временем жизни, лишь усугубило их.

Как правило, объектно-ориентированная обработка ошибок подразумевает наличие виртуального метода Error, которому передается управление в случае нештатных ситуаций, возникших при выполнении методов этого объекта. Иногда в состав объекта включают также поле состояния, в котором фиксируется информация об ошибке. Подобная техника использована в типе TStream и его наследниках (Turbo Vision).

Наибольшие проблемы вызывает ситуация, когда внешняя процедура обращается к статическому методу объекта, который, в свою очередь, обращается к виртуальному методу. Если последний был переопределен таким образом, что в некоторых случаях при его выполнении может возникнуть нештатная ситуация, не предусмотренная в родительском объекте, то ее обработка становится проблематичной. В таких ситуациях описанный выше механизм исключений может оказаться незаменимым.

Специальная техника применяется для обработки ошибок при конструировании объекта. Если сконструировать объект не удается, то следует выполнить встроенную процедуру fail (ей можно пользоваться только внутри конструктора.) Fail, подобно exit, осуществляет возврат, однако если эта процедура вызвана для объекта, размещаемого с помощью оператора new в динамической памяти, то память, выделенная для конструирования объекта, будет освобождена. Вызов конструктора может быть оформлен как вызов булевской функции, true возвращается в случае успешного создания объекта, false — если из конструктора была вызвана процедура fail.

Описанной техники иногда бывает недостаточно, когда в вызывающей программе необходимо знать точную причину неудачного конструирования. В таких случаях можно воспользоваться кодом возврата в виде var-параметра или глобальной переменной.

Заключение

Выбор способа обработки исключительных ситуаций остается сложной проблемой. Обычно не удается пользоваться только одним из них, однако применение их всех вместе в одной программе снижает ее читабельность и надежность. Наиболее важная функция обработчиков исключений — восстановление после ошибки. Введение входного контроля и правильная организация данных (особенно хранимых) значительно сокращают потребность в восстановительных работах, но редко удается избежать их совсем.

Хотелось бы думать, что фирма Borland обогатит Turbo Pascal стандартным средством обработки исключительных ситуаций. В языке C++ для таких средств уже зарезервированы ключевые слова try, catch и throw, и можно предположить, что они будут реализованы в одной из следующих версий компилятора. ■

Autodesk.

Autodesk RF (СП "Параллель") - Российский филиал американской компании Autodesk, Inc. - продолжает продажи лицензионного программного обеспечения (AutoCAD, Autodesk 3D Studio, Autodesk Animator Pro) и приглашает к сотрудничеству дилеров, разработчиков, агентов, учебные центры.

В связи с принятием российским парламентом законов "Об авторском праве и смежных правах", "О правовой охране программ для электронных вычислительных машин и баз данных" Autodesk RF напоминает, что нелегальное распространение и использование программной продукции компании Autodesk приведёт к конфликтным ситуациям во взаимоотношениях с деловыми партнерами.

Autodesk RF invites a marketing manager and a sales engineer. Requirements: CAD background, AutoCAD knowledge, native Russian, fluent English, ability to travel. Applicants should mail the CV to:

Autodesk RF, 107005, Москва, 2 Бауманская ул., 9/23, корп. 18. Тел. (095) 261 6363, факс 267 9927

Анонс

«Монитор» — версия 6.93

А. Штрик. Языки программирования четвертого поколения. Часть 1
Рассказывается о причинах, приведших к появлению языков 4GL; описываются основные черты языков программирования этого класса.

А. Колесов. Профессиональная работа в среде Microsoft BASIC. Часть 3
Рассматриваются проблемы, возникающие при создании больших программ, даются практические рекомендации по минимизации размеров исполняемых модулей и сокращению времени компиляции.

Е. Морозов, Е. Федорова. Средства сетевых СУБД для восстановления баз данных
На примерах конкретных сетевых СУБД рассматриваются средства восстановления баз данных после отказов.

Е. Кисель. Эффекты для демонстрации слайдов
В общем виде постановку задачи можно сформулировать так: мы хотим научиться писать программы, отображающие на экране последовательности слайдов с использованием эффектов.

М. Трофимов. Специфические ошибки Turbo Pascal
Программируя на Turbo Pascal, легко сделать ряд трудно обнаруживаемых ошибок, в то время как более старые компиляторы позволяют их избежать. В статье рассмотрен ряд типичных ошибок и обсуждаются некоторые методы борьбы с ними.

Листинг 1. Реализация механизма исключений на Turbo Pascal

```
unit Excs;
{*************************************************  *}
{*                  Реализация механизма исключений  *}
{*                  Дм. Арапов 1993г.                *}
{*************************************************  *}
interface
const MaxLevel = 64;    {* Максимальная вложенность областей обра-*}
                        {* ботки исключений. При необходимости     *}
                        {* можно изменить эту константу.           *}
type TExc = word;       {* Тип кода исключительной ситуации.       *}
                        {* Значения от 0 до 64 зарезервированы.    *}
{*************************************************  *}
{* Прежде чем использовать исключение, его следует зарегист-*}
{* рировать с помощью процедуры ExcRegister. Регистрационная *}
{* запись должна находиться в сегменте данных и содержать поле *}
{* кода исключения и смещение диагностического сообщения, ко-*}
{* торое также должно находиться в сегменте данных.          *}
{* Значение поля Next устанавливается процедурой регистрации. *}
{*************************************************  *}
    PExcRec = ^TExcRec;
    TExcRec =
        record
            Code    : TExc;
            Msg     : word;
            Next    : word;
        end;
procedure ExcRegister( var R : TExcRec );
const   exOk = 0;       {* Код нормального выполнения.          *}
var     RC : TExc;      {* Переменная RC после возбуждения исключе-*}
                        {* ния содержит код исключительной ситуации.*}
function  ExcBegin : TExc;
{* Вход в область обработки исключительных ситуаций. Возвра- *}
{* щает значение exOk. При возникновении исключительной си- *}
{* туации управление вновь попадает внутрь этой функции. В  *}
{* этом случае она возвращает код исключения.                *}
procedure ExcEnd;
{* Выход из области обработки исключительных ситуаций.      *}
procedure Raise( E : TExc );
{* Возбуждение исключения. Управление передается внутрь по- *}
{* следней вызванной функции ExcBegin. Переменной RC присва- *}
{* ивается код исключения. При вызове вне области обработки  *}
{* исключений выдает диагностическое сообщение и прекращает  *}
{* выполнение программы.                                     *}
{*************************************************  *}
{*          Регистрационные константы исключений модуля Excs *}
{*************************************************  *}
const   emExcUnknown    : String[17] = 'Unknown exception';
        emExcOverflow   : String[24] = 'Exception stack overflow';
        emExcUnderflow  : String[25] = 'Exception stack underflow';
        emExcRegister   : String[28] = 'Exception registration error';
const   erExcOverflow   : TExcRec = ( Code : $0001;
                                      Msg : Ofs(emExcOverflow) );
const   erExcUnderflow  : TExcRec = ( Code : $0002;
                                      Msg : Ofs(emExcUnderflow) );
const   erExcRegister   : TExcRec = ( Code : $0003;
                                      Msg : Ofs(emExcRegister) );
implementation
var ExcRecList: word; {* Голова списка регистрационных записей *}
type TExcLevel      = record OldSP6, OldBP, OldIP, OldCS : word end;
var  ExcStackTop    : word;
var  ExcStackEnd    : word;
var  ExcStack       : array[ 0..MaxLevel-1 ] of TExcLevel;
type PString        = ^String;
procedure Finish; near;
var Temp : word;
begin { Finish }
    Temp := ExcRecList;
    while (Temp <> 0) and (PExcRec( Ptr(DSeg,Temp))^.Code <> RC ) do
        Temp := PExcRec( Ptr(DSeg,Temp))^.Next;
    if Temp = 0
        then WriteLn( emExcUnknown )
    else WriteLn( PString( Ptr( DSeg,PExcRec( Ptr(DSeg,Temp))^.Msg))^ );
    Halt( Lo( Rc ));
end; { Finish }
procedure ExcRegister;
begin { ExcRegister }
    if Seg( R ) <> DSeg then Raise( erExcRegister.Code );
    R.Next := ExcRecList;
    ExcRecList := Ofs(R);
end; { ExcRegister }
function  ExcBegin; assembler;
asm  { ExcBegin }
    Push    BP
    Mov     BP,SP
    Mov     DI,[ ExcStackTop ]
    Cmp     DI,[ ExcStackEnd ]
    Je      @@1
    Mov     AX,DS
    Mov     ES,AX
    Mov     AX,BP
    Stosw
    Mov     AX,[BP]
    Stosw
    Mov     AX,[BP+2]
    Stosw
    Mov     AX,[BP+4]
    Stosw
    Mov     [ ExcStackTop ],DI
    Xor     AX,AX
    Mov     SP,BP
    Pop     BP
    Retf
@@1:
    Mov     AX,[erExcOverflow.Code]
    Mov     SP,BP
    Pop     BP
    Retf
end; { ExcBegin }
procedure ExcEnd; assembler;
asm  { ExcEnd }
    Mov     DI,[ExcStackTop]
    Lea     AX,[ExcStack]
    Cmp     DI,AX
    Je      @@2
    Sub     DI,8
    Mov     [ExcStackTop],DI
    Retf
@@2:
    Mov     AX,[erExcUnderflow.Code]
    Mov     [RC],AX
    Call    near ptr Finish
end; { ExcEnd }
procedure Raise; assembler;
asm  { Raise }
    Mov     AX,[BP+6]
    Or      AX,AX
    Jz      @@3
    Mov     [RC],AX
    Mov     SI,[ExcStackTop]
    Lea     DX,[ExcStack]
    Cmp     SI,DX
    Je      @@4
    Sub     SI,8
    Mov     [ExcStackTop],SI
    Mov     SP,[SI]
    Mov     BP,SP
    Mov     DX,[SI+2]
    Mov     [BP],DX
    Mov     DX,[SI+4]
    Mov     [BP+2],DX
    Mov     DX,[SI+6]
    Mov     [BP+4],DX
@@3:
    Pop     BP
    Retf
@@4:
    Call    near ptr Finish
end; { Raise }
{*************************************************  *}
{* Инициализация механизма исключений состоит из инициализации ре-*}
{* гистрационного списка и инициализации стека областей обработки. *}
{*************************************************  *}
begin
    ExcStackTop := Ofs( ExcStack );
    ExcStackEnd := ExcStackTop + MaxLevel*SizeOf(TExcLevel);
    ExcRecList  := 0;
    ExcRegister( erExcOverflow );
    ExcRegister( erExcUnderflow );
    ExcRegister( erExcRegister );
end. { Excs }
```

Листинг 2

```
program TestExcs;
{*************************************************  *}
{*      Тест на использование аппарата исключительных ситуаций *}
{*                                  Дм.Арапов 1993г.  *}
{*************************************************  *}
uses Excs;
const   emMyError   : String[10] = 'User error';
const   erMyError   : TExcRec = ( Code : $0100; Msg : Ofs( emMyError ));
procedure ContainError;
begin
    Raise( erMyError.Code );
end;
begin
    ExcRegister( erMyError );
    case ExcBegin of
        0 : begin
                ContainError;
                ExcEnd;
            end;
        else begin
                Raise( Rc );
            end;
    end; { case }
end. { TestExcs }
```

ELITSoft

Пользователям вычислительной техники Акционерное общество "ЭЛИТ СОФТ" предлагает

Агрегаты Бесперебойного Питания

(Uninterruptable Power Source)

Основные характеристики

- Мощность 900 ВА
- Время автономной работы 40 мин.
- Время переключения 5-7 мсек.
- Рабочий диапазон 180-240 В
- Гарантия 1 год
- Послегарантийное обслуживание

В конструкции предусмотрены

- Фильтрация сетевых помех
- Звуковая и световая сигнализация
- Индикация перегрузки
- Индикация уровня заряда батарей
- Герметичные, необслуживаемые аккумуляторные батареи

Поставка и оплата

Поставка со складов в Москве, Санкт-Петербурге, Чите, Рязани.
Любая удобная Вам форма оплаты.

Адрес: 105215 Москва, 11-я Парковая, 44, корп. 1
Телефон:
секретариат: (095) 965-35-29, 532-96-14
отдел сбыта: (095) 965-35-28, 965-35-30
Факс (095) 965-35-28, 965-35-30
Телетайп 205211 КТУР

Мы продаем компьютеры!

SYMANTEC

TIME LINE

FOR WINDOWS ®

Русскоязычная версия!

Самая современная технология управления проектами...
Ноу-хау в области принятия решений...
Теория и практика американского менеджмента...
Все это будет доступным для Вас, если Вы станете пользователем Time Line для Windows!

Диалог-МИФИ
095-320-3466/F324-3055

Перспективные технологии
095-256-4473/F259-2727

Merisel-Russia
095-276-9008/F274-0097

Трио Плюс
095-971-1204/F281-5236

Turbo Vision: обнаружение некоторых ошибок в программах

В. Захаров,
И. Кузнецов

Каждый человек, избравший своим профессиональным занятием разработку программного обеспечения для ЭВМ, рано или поздно, каких бы высот мастерства в этой области он ни достиг, осознает неопровержимость неумолимого закона программирования: когда в программе исправлена последняя ошибка, она становится никому не нужной.

И хотя авторы статьи всегда готовы в который раз после очередной «поломки» какой-либо программы в совершенно неподходящий момент, когда создается новый программный шедевр, в сердцах произнести перед недоумевающим заказчиком эту производящую впечатление на непрофессионала фразу, перед тем как кинуться искать в своем детище «предпоследнюю» ошибку, они еще не потеряли надежды пусть не сейчас, а через какое-то время взять и научиться создавать нужные кому-то программы, более сложные чем 2x2, в которых не будет ошибок. А кому из вас, господа Программисты, этого не хочется? Всем хочется, но кто может?

Поскольку мы не волшебники, а все еще учимся, то речь мы поведем не о потрясающих воображение чудесах, позволяющих создавать безошибочные программные системы, а о небольших, но, на наш взгляд, полезных приемчиках «колдовства», помогающих не пропустить в программах некоторые досадные ошибки.

Коррекция модуля HelpFile

Одним из последних достижений программирующего человечества в борьбе за чистоту программных средств от ошибок является объектно-ориентированная техника создания программ, реализованная, в частности, в расширении такого языка массового использования, как Pascal, в программном продукте фирмы Borland Turbo Pascal 6.0.

Как известно, Turbo Pascal 6.0, кроме техники объектно-ориентированного программирования, предоставляет в распоряжение программиста объектно-ориентированную оболочку Turbo Vision для создания диалоговых многооконных программ.

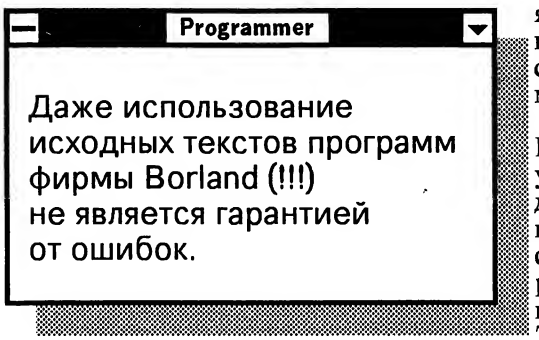
Авторы статьи уже более двух лет успешно используют возможности Turbo Vision для создания программного обеспечения как в области высокоэффективных систем обработки информации с применением TACCESS и Paradox Engine 2.0, так и в области автоматизации проектирования радиоэлектронной аппаратуры. Однако работа с Turbo Vision еще раз подтвердила набившую оскомину истину — ошибки, ошибки и только ошибки.

Turbo Vision имеет встроенный инструмент организации контекстно-ориентированной подсказки в прикладных программах, состоящий из компилятора файлов подсказки TVHC и модуля HelpFile, включаемого в разрабатываемую программу и обеспечивающего появление и удаление окна с контекстной подсказкой.

Заметим, что в фирменном руководстве по Turbo Vision описание организации в прикладной программе контекстно-ориентированной подсказки практически отсутствует и пользователю предлагают рассмотреть пример организации подсказки, включенный в дистрибутивные диски Turbo Pascal. Этот пример состоит из программы TVDemo и модуля HelpFile.

После рассмотрения примера Вы в течение получаса вставляете в свою программу модуль HelpFile, вызов подсказки при нажатии клавиши F1, создаете файл подсказок, транслируете и, казалось бы, получаете достаточно удобный программный продукт, который удовлетворит любого капризного заказчика. Все происходит очень быстро и слишком хорошо, при столь малых затратах сил, что даже не верится. И правильно, что не верится. «Полтергейст» начнется

чуть позже, возможно, только на следующий день или даже через неделю. Трудно с точностью до минуты предсказать, когда Ваша программа, дополненная средствами контекстной подсказки, после очередного нажатия на клавишу F1 приведет к полному «зависанию» Вашего компьютера или компьютера заказчика, но то, что это обязательно произойдет, гарантировать можно на 100%. Причем сначала это будет воспринято как сбой компьютера или диска, или чего-то там еще, только не Вашей программы — Вы же использовали исходные тексты программы А-МЕ-РИ-КАН-СКОЙ фирмы Borland, тем более, что, заново оттранслировав программу, Вы увидите, что все работает вроде бы правильно. Но программирование и в Америке программирование. Рано или поздно Вы убедитесь, что компьютер «зависает» по вине Вашей программы.



В такой ситуации естественно начать с исходного этапа и посмотреть, почему же такое не происходит с программой TVDemo из Turbo Pascal? И если Вы проявите немного терпения, то обнаружите, что и TVDemo — демонстрационная программа возможностей Turbo Vision — также приводит к «зависанию» компьютера. Чтобы убедиться в этом, необходимо осуществить полную перезагрузку Вашего компьютера, после чего сразу же стартовать программу TVDemo.exe. А теперь нажимайте поочередно на клавиши F1 и Esc, и приблизительно через минуту Ваш компьютер «зависнет» так, что даже нажатие на Ctrl-Alt-Del не приведет его в работоспособное состояние.

Если, проделывая эту процедуру, Вы будете внимательно наблюдать за изменениями в строке статуса программы TVDemo, то увидите, что, в добавление ко всему, после каждого просмотра подсказки происходит уменьшение размера свободной динамически распределяемой памяти (Heap Turbo Pascal 6.0) пропорционально объему текста подсказки. Это сразу же наводит на мысль, что в модуле HelpFile имеются объекты или переменные, которые не освобождают динамически распределяемую память после ее использования.

Итак, наличие ошибок установлено, теперь остается только найти способы их устранения.

Самое простое, с чего можно начать в этой ситуации, — это определить объекты или переменные в модуле HelpFile, которые не освобождают память после ее использования. Действительно, в результате анализа текста модуля HelpFile оказывается, что объекты THelpTopic и THelpIndex не содержат деструкторов THelpTopic.Done и THelpIndex.Done, перекрывающих деструктор TObject.Done, обращение к которым происходит через процедуру Dispose в деструкторах THelpFile.Done и THelpViewer.Done.

Для устранения этой ошибки необходимо в описание объектов THelpTopic и THelpIndex добавить описание деструкторов, то есть вставить в текст объявления каждого из этих объектов строку

```
destructor Done; virtual;
```

и дополнить тексты методов этих объектов текстами двух деструкторов, приведенных в листинге 1.

Перетранслировав программу TVDemo после этих добавлений, Вы сможете убедиться, что после завершения просмотра текста подсказки не происходит уменьшения свободной динамически распределяемой памяти программы. Хотя эта ошибка не является критичной для TVDemo, но для Вашей программы, решающей более сложную задачу, чем простой просмотр файлов, и требующей соответственно больше памяти, она может оказаться весьма неприятной.

Следует заметить, что отсутствие освобождения динамически распределяемой памяти после окончания ее использования является довольно распространенной ошибкой, и как ее избежать в общем случае, мы рассмотрим немного позднее.

А сейчас вернемся к модулю HelpFile, поскольку устранение уменьшения динамически распределяемой памяти при просмотре подсказок не избавляет от непредсказуемого «зависания» компьютера во время выполнения Вашей программы или программы TVDemo. В результате поиска места возникновения этой ошибки удалось установить, что «зависание» компьютера при работе модуля HelpFile происходит во внутренней процедуре TextToLine метода WrapText объекта THelpTopic. Сокращенный исходный текст метода WrapText, необходимый для понимания дальнейших рассуждений, представлен в листинге 2. «Зависание» компьютера, однако, происходит не из-за наличия каких-либо ошибок в процедуре TextToLine, а вследствие вызова этой процедуры с отрицательным значением третьего фактического параметра (I : integer), подставляемого вместо формального параметра Length этой процедуры.

Причиной этого явления оказывается превышение диапазона представления целых положительных чисел типа integer в функции Scan, в результате чего значение функции Scan становится отрицательным.

После установления этого факта оказывается, что устранить ошибку возможно несколькими способами, один из них мы приводим в листинге 3.

К сказанному следует добавить, что авторы в течение года держали под контролем работу скорректированного модуля HelpFile в ряде прикладных программ с различными файлами контекстной подсказки и за этот период ни одного случая «зависания» компьютера при просмотрах на экране монитора контекстной подсказки обнаружено не было.

Контроль освобождения динамически распределяемой памяти

Одной из распространенных ошибок при составлении программ является неаккуратное обращение с динамически распределяемой памятью, в частности, «забывание» освободить ее после использования, что в конце концов может привести к аварийному завер-

шению программы из-за нехватки оперативной памяти (ОП). Поиск же мест «засорения» ОП в отлаживаемых программах, занимающих значительный объем, может отнять много времени, поскольку к моменту комплексной отладки, когда эти ошибки попадаются на глаза, естественным образом забываются многие детали создания программных компонентов. Поэтому целесообразно с самого начала отладки программ иметь средство индикации занятости динамически распределяемой памяти после окончания ее использования.

Для этого требуется совсем немного: определить размер свободной области ОП в начале и конце работы программы и в случае неравенства выдать соответствующее сообщение.

Если Вы работаете в Turbo Pascal 6.0, то в этом Вам может помочь объект TMemoryTest (пример использования дан в листинге 4, описание в листинге 5).

Необходимо заметить, что при работе с Turbo Vision рекомендуется поместить вызов конструктора объекта TMemoryTest в конструкторе потомка объекта TApplication после оператора TApplication.Init, а вызов деструктора TMemorytest в соответствующем деструкторе перед вызовом TApplication.Done. Иначе не будет отслежена ситуация, когда не были своевременно удалены видимые элементы Turbo Vision.

Возможно создание нескольких экземпляров объекта TMemoryTest, когда желательно более точно определять места «засорения» ОП. В этом случае полезно задавать в конструкторе объекта строку-идентификатор, попадающую в сигнальное сообщение.

Использование предложенного объекта наиболее полезно с самого начала отладки программы, когда более просто установить место для «потерянных» операторов освобождения памяти. Отладив фрагменты программы во всевозможных режимах (имея в виду аккуратность их обращения с ОП), можно исключить их анализ при возникновении ситуации «засорения» во всей программе.

Итак, если Вы потрудитесь над вставкой объекта TMemoryTest своевременно, то сможете предупредить многие некорректности в процессе отладки и модификации Вашей программы.

Объект для демонстрации параметров работы циклической программы

Когда приходится ожидать окончания какой-либо достаточно продолжительной обработки, психологически оправдано обеспечить программу признаками ее жизни и, если возможно, элементами прогнозирования общего времени обработки. Подобные средства могут быть полезны и при отладке для определения моментов зацикливания и грубой оценки быстродействия участков программ.

Когда программа имеет дело с достаточно протяженным циклом (например, при сканировании записей БД) и известно число прохождений цикла, возможна индикация таких переменных, как процент выполненной обработки, текущее и максимально возможное значение переменной цикла.

Полезно и визуально отслеживать (особенно при отладке) время обработки и текущий размер оперативной памяти.

Подобную задачу помогает решить объект TRunningLine (листинг 5), позволяющий, в зависимости от задаваемых при его инициализации опций, выводить в окно Turbo Vision следующие параметры:
- бегущую строку, подобную той, которая выдается в Norton Commander при копировании и перемещении файлов,
- текущее значение переменной цикла,
- максимально допустимое значение переменной цикла,
- процент завершения обработки,
- время обработки,
- размер свободной области ОП.

Пример получаемого в результате работы объекта TRunningLine окна показан на рис. 1.

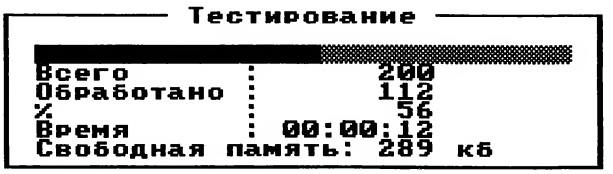


Рис. 1

При вызове конструктора объекта Init требуется задать опции отображения с помощью константы rlXXXXX (листинг 4). В этом случае окно автоматически центрируется. Для задания координат окна возможно применение конструктора InitRect, дополнительным параметром которого является объект TRect. Чтобы не потерять выводимые строки, следует соблюдать равенство

R.B.Y — R.A.Y — 3 =
= число отображаемых строк

Максимальное количество строк равно 6.

Метод StatusDisplay, имеющий параметр, задающий значение переменной цикла, вызывает изменение состояния окна. При завершении отслеживаемой обработки вызывается метод Done, удаляющий объект TRunningLine.

Если время выполнения цикла мало, а число прохождений цикла велико, то обращение к методу StatusDisplay при каждой отработке цикла может вызвать увеличение времени работы программы. Это объясняется не только тем, что метод использует недостаточно эффективный код (например, не обращается прямо в видеопамять), любой, пусть даже «обезжиренный» добавочный код, требует времени на свое выполнение, особенно если он работает с экраном монитора. Поэтому рекомендуем в таком случае вызывать StatusDisplay через определенное количество прохождений цикла. В примере, приведенном в листинге 4, вызов StatusDisplay производится один раз за 16 выполнений цикла.

Итак, объект TRunningLine позволяет:

1) прогнозировать время выполнения фрагмента программы,

2) визуально отслеживать состояние свободной области ОП, что помогает выявлять некоторые ошибки управления памятью,

3) оценивать быстродействие программы. ■

Листинг 1. Тексты деструкторов THelpTopic.Done и THelpIndex.Done

```
destructor THelpTopic.Done;
var
    I, Size: Integer;
    PP: PParagraph;
    DD: PParagraph;
begin
    PP := Paragraphs;
    while PP <> nil do
    begin
        DD := PP;
        PP := PP^.Next;
        FreeMem(DD, SizeOf(DD^) + DD^.Size);
    end;
    if NumRefs > 0 then
        FreeMem(CrossRefs, NumRefs * SizeOf(TCrossRef));
end; { THelpTopic.Done }

destructor THelpIndex.Done;
begin
    if Size > 0 then FreeMem(Index, Size * SizeOf(Longint));
end; {THelpIndex.Done}
```

Листинг 2. Сокращенный исходный текст метода WrapText объекта THelpTopic

```
function THelpTopic.WrapText(var Text; Size: Integer;
                             var Offset: Integer; Wrap: Boolean): String;

    ..............

var
    Line: String;
    I, P: Integer;

    ..............

function Scan(var P; Offset: Integer; C: Char): Integer; assembler;
asm
    CLD
    LES     DI,P
    ADD     DI,&Offset
    MOV     CX,0FFFFH
    MOV     AL, C
    REPNE   SCASB
    INC     CX
    NEG     CX
    XCHG    AX,CX
end; { Scan }

procedure TextToLine(var Text; Offset, Length: Integer; var Line: String);
    assembler;
asm
    CLD
    PUSH    DS
    LDS     SI,Text
    ADD     SI,&Offset
    LES     DI,Line
    MOV     AX,Length
    STOSB
    XCHG    AX,CX
    REP     MOVSB
    POP     DS
end; { TextToLine }

begin
    I := Scan(Text, Offset, #13);
    if I + Offset > Size then I := Size - Offset;
    if (I >= Width) and Wrap then
    begin

        ..............

    end;
    TextToLine(Text, Offset, I, Line);
    if Line[Length(Line)] = #13 then Dec(Line[0]);
    Inc(Offset, I);
    WrapText := Line;
end; { THelpTopic.WrapText }
```

Листинг 3. Один из способов исправления метода WrapText объекта THelpTopic

```
function THelpTopic.WrapText(var Text; Size: Integer;
                             var Offset: Integer; Wrap: Boolean): String;
var
    Line : String;
    I, P  : Integer;
    Kz   : word;          { Добавления авторов статьи }

    ..............

function Scan(var P; Offset: Integer; C: Char): word; assembler;
                    { Изменен тип функции Scan }
asm

    ..............

end; { Scan }

    ..............

begin
{ Изменения авторов статьи }
    Kz := Scan(Text, Offset, #13);
    if Kz + Offset > Size then I := Size - Offset else i := Kz;
{ Конец изменений авторов статьи }

    ..............

end; { THelpTopic.WrapText }
```

Листинг 4. Пример использования объектов TMemoryTest и TRunningLine

```
program PasTst01;
uses    Crt, App, PasTools;
const   imax = 200;
var     i           : word;
        TApp        : TApplication;
        prl         : pRunningLine;
        pmt1,pmt2   : pMemoryTest;
begin
    TApp.Init;
    New(pmt1,Init('pmt1'));
    New(pmt2,Init('pmt2'));
    New(prl,Init('Тестирование',iMax,rlFullInf));
    for i := 1 to iMax do
    begin
        if  i  AND $000f = 0
            then prl^.StatusDisplay(i);
        Delay(100);
    end;
    Dispose(prl,Done);
    Dispose(pmt1,Done);     {Выдается сообщение о засорении ОП,
                             так как не удален экземпляр pmt2^
                             объекта TMemoryTest}
    TApp.Done;
end.
```

Листинг 5. Описание объектов TMemoryTest и TRunningLine

```
unit pastools;

interface

uses Dos, App, Objects, Drivers, Views ;

const
    rlRunningLine = $01;   {Флаг выдачи бегущей строки}
    rlMaxCount    = $02;   {Флаг выдачи максимального значения
                            переменной цикла}
    rlCurrCount   = $04;   {Флаг выдачи текущего значения
                            переменной цикла}
    rlPercent     = $08;   {Флаг выдачи доли прошедшей обработки}
    rlExecTime    = $10;   {Флаг выдачи времени обработки}
    rlMemAvail    = $20;   {Флаг выдачи размера свободной памяти}
    rlFullInf     = rlRunningLine OR rlCurrCount OR rlMaxCount OR
                    rlPercent OR rlMemAvail OR rlExecTime;

type
    pRunningLine  = ^TRunningLine;
    TRunningLine  = object(TWindow)

constructor Init(ATitle : string; MaxCount : longint; AOptions : byte);
constructor InitRect(R: TRect; ATitle : string;
                     MaxCount : longint; AOptions : byte);

procedure StatusDisplay(CurrCount : longint);
    private
        pInterior : pView;
    end;

    pMemoryTest = ^TMemoryTest;
    TMemoryTest = object {проверка освобождения ОП}
```

Листинг 5 (продолжение)

```
constructor Init(ATitle : string) ;
destructor Done;
procedure ErrDisplay(Size : longint);
    private
        MemAvailStart      : longint;
        Ident              : string[8];
    end;

implementation

type
    pRunningLineInterior    = ^TRunningLineInterior;
    TRunningLineInterior    = object(TView)
constructor    Init(RBounds : TRect; MaxCount : longint; AOptions : byte);

procedure      Draw; virtual;

procedure      SetCurrentCount(CurrCount : longint);
    private
        rlOptions          : byte;
        CurrentCount       : longint;
        MaximumCount       : longint;
        Time               : longint; { в секундах }
    end;

{---------------------------- TRunningLine ----------}

constructor TRunningLine.Init (ATitle : string;
                               MaxCount : longint; AOptions : byte);

var
    R     :TRect;
    nr,i  : byte;
begin
    nr := 0;
    for i := 0 to 5 do
        if (AOptions AND ($0001 SHL i)) <> 0 then Inc(nr);
    R.Assign(0,0,32,nr+3);
    R.Move((Desktop^.Size.X - R.B.X) div 2, (Desktop^.Size.Y - R.B.Y) div 2);
    InitRect(R, ATitle, MaxCount,AOptions);
end; {TRunningLine.Init}

constructor TRunningLine.InitRect
    (R: TRect; ATitle : string; MaxCount : longint; AOptions : byte);
begin
    TWindow.Init(R,ATitle,wnNoNumber);
    Options := Options AND (NOT ofSelectable);
    GetExtent(R);
    R.Grow(-1,-1);
    pInterior := New(pRunningLineInterior,Init(R,MaxCount,AOptions));
    Insert(pInterior);
    DeskTop^.Insert(@Self);
end; {TRunningLine.InitRect}

procedure TRunningLine.StatusDisplay(CurrCount : longint);
var
    pRL : pRunningLineInterior absolute pInterior;
begin
    pRL^.SetCurrentCount(CurrCount);
    pRL^.DrawView;
end; {TRunningLine.StatusDisplay}

{---------------------------- TRunningLineInterior ----------}

constructor    TRunningLineInterior.Init(RBounds : TRect;
               MaxCount : longint; AOptions : byte);
var  h,m,s,s100 : word;
begin
    TView.Init(RBounds);
    rlOptions          := AOptions;
    MaximumCount       := Abs(MaxCount);
    CurrentCount       := 0;
    GetTime(h,m,s,s100);
    Time               := (h*60 + m)*60 + s;
end; {TRunningLineInterior.Init}

procedure TRunningLineInterior.Draw;
var  cs : string;

procedure FillTime;
var
    h,m,s,s100     : word;
    et             : longint;
    Param          : record h,m,s : longint; end;
begin
    GetTime(h,m,s,s100);
    et := (h*60 + m)*60 + s - Time;
    if et < 0 then Inc(et,24*60*60);
    with Param do begin
        s     := et mod 60;
        m     := (et div 60) mod 60;
        h     := et div 3600;
    end;
    FormatStr(cs,'%02d:%02d:%02d',Param);
    cs := 'Время                : ' + cs;
end;

procedure CountToLine(Count : longint; ATitle : string);
begin
    Str(Count:8,cs);
    cs := '  ' + ATitle + ':' + cs;
end;

const
    rlFirstLine = $80;
    rlMass : array[0..6] of byte =
                        (rlFirstLine,rlRunningLine,rlMaxCount,rlCurrCount,
                        rlPercent,rlExecTime,rlmemAvail);
var
    Percent,i,qCols    : byte;
    Color              : word;
    nr                 : integer;
    B                  : TDrawBuffer;
begin
    if MaximumCount = 0
        then Percent := 0
    else Percent := CurrentCount * 100 div MaximumCount;
    qCols := Percent * (Size.X-2) div 100;
    Color := GetColor(1);
    nr := -1;
    for i := 0 to 6 do begin
        cs := '';
        case ((rlOptions OR rlFirstLine) AND rlMass[i]) of
            rlFirstLine     : cs := '  ';
            rlRunningLine : begin
                while Length(cs) < qCols
                    do cs := cs + #219;
                while Length(cs) < Size.X-2
                    do cs := cs + #177;
                cs := '  ' + cs + '  ';
            end;
            rlMaxCount    : CountToLine(MaximumCount,'Всего          ');
            rlCurrCount   : CountToLine(CurrentCount,'Обработано   ');
            rlPercent     : CountToLine(Percent          ,'%          ');
            rlExecTime    : FillTime;
            rlMemAvail    : begin
                Str((MemAvail div 1024):3,cs);
                cs := 'Свободная память: ' + cs + ' кб';
            end;
        end; {case}
        if Length(cs) <> 0 then begin
            Inc(nr);
            while Length(cs) < Size.X do cs := cs + ' ';
            MoveStr(B,cs,Color);
            WriteLine(0,nr,Size.X,1,B);
        end;
    end; {for cycle}
end; {TRunningLineInterior.Draw}

procedure TRunningLineInterior.SetCurrentCount(CurrCount : longint);
begin
    CurrentCount := Abs(CurrCount) mod (MaximumCount+1);
end; {TRunningLineInterior.SetCurrentCount}

{---------------------- TMemoryTest ------------}

constructor TMemoryTest.Init(ATitle : string) ;
begin
    MemAvailStart      := MemAvail;
    Ident              := ATitle;
end; {TMemoryTest.Init}

destructor TMemoryTest.Done;
var
    i : longint;
begin
    i := MemAvailStart - MemAvail;
    if i <> 0 then ErrDisplay(i);
end; {TMemoryTest.Done}

procedure TMemoryTest.ErrDisplay(Size : longint);
begin
    WriteLn(Ident + ': не освобождена ОП размером (в байтах) =',Size);
    Writeln('Нажмите "Enter"');
    Readln;
end;            {TMemoryTest.ErrDisplay}

end.
```

Российское научно–техническое общество РЭС им. А.С.Попова Министерства связи России приглашает фирмы и организации принять участие в крупнейшей выставке года — Международной выставке–ярмарке программных и аппаратных средств, которая состоится в Москве с 5 по 9 октября 1993 года на территории ВВЦ (бывшая ВДНХ). Выставку проводит МП «Экспосервис».

Справки по тел. (095) 921-06-59. Факс: (095) 924-70-72.

МАГАЗИН
+LINK

ЭТО ВСЁ ДЛЯ КОМПЬЮТЕРНЫХ СЕТЕЙ И КОМПЬЮТЕРОВ

Тел. 263-94-40 *Факс 267-58-80*
Ново-Басманная ул., 20

Вы можете приобрести в нашем магазине любое оборудование к Вашему компьютеру:

- Сетевые адаптеры ArcNet, EtherNet.
- Модемы и факс-модемы от 2400 до 14400.
- Картриджи для матричных принтеров.
- Системы бесперебойного питания UPS APC.
- Комплектующие к ПЭВМ.
- Компьютеры.
- Стриммеры.
- Сканеры и прочее.

Мы ждем Вас!

Восстановление красящих лент для принтеров.

Телефон 238–99–33.

ТИВИОНИКА

Все для студий компьютерной графики

от плат видеовывода до суперкомпьютеров

Silicon Graphics

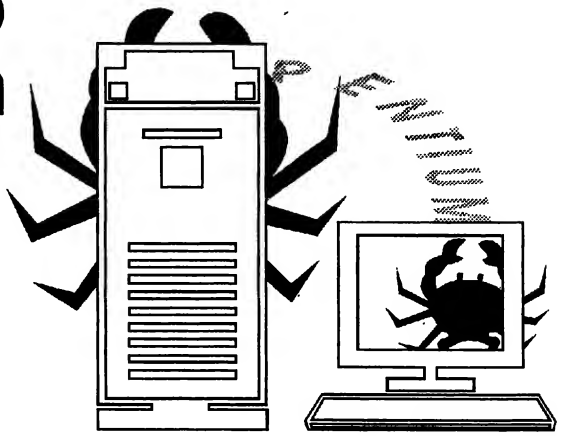
тел: (095) 171-7778
факс: (095) 365-0208

Журнал «Монитор» можно приобрести в городах России, Украины, Беларуси, Узбекистана, обратившись к нашим официальным дилерам.

1. Самара, МП «Сети», директор Дмитрий Грецов
Тел. (8462) 354-932
2. Волгоград, арендное предприятие «Техническая книга»
Тел. (8442) 363-597
3. Чебоксары, фирма «Демос»
Тел. (8352) 208-395, 426-492, факс 209-437
4. Киев, МП «Диас»
Тел./факс (044) 220-77-15, тел. 518-80-24
5. Днепропетровск, Попов А.Е.
Тел. (0562) 428-593
6. Минск, НПП «Триумф»
Тел. (0172) 661-609, 661-680, факс 666-606
7. Ташкент, Центр молодежных исследований
Тел. (3712) 688-847
8. Тамбов, АО «От Елены» ЛПС «Shift»
Тел. (0752) 22-44-63
9. Пермь, Попов Андрей Леонидович
Тел. (3422) 33-30-91
10. Иркутск, Мадьяров Игорь Иосифович
Тел. (3952) 46-19-68

Микропроцессор Pentium

Д. Бройтман



Часть 3
Численная арифметика

В состав Pentium входит высокопроизводительная подсистема поддержки вычислений над действительными числами, целыми числами большой размерности и двоично кодируемыми десятичными числами. Эти функции выполняет устройство для операций с плавающей точкой (Floating-point Unit — FPU). Pentium FPU полностью перепроектировано по сравнению с Intel 486 FPU. Pentium FPU поддерживает стандарты IEEE 754 и 854 для арифметики с плавающей точкой и совместимо на уровне объектного кода с математическими сопроцессорами 8087, Intel 287, Intel 387DX, Intel 387SX, Intel 487DX и с процессором Intel 486DX.

1. Производительность

Реальная производительность зависит от характеристик решаемой прикладной задачи. Ориентировочное соотношение скоростей выполнения основных операций с плавающей точкой для 66 МГц Pentium и 66 МГц Intel486 DX2 приведено в таблице 1.

Производительность и пропускная способность систем на основе Pentium также может быть повышена за счет использования возможности Pentium одновременно выполнять целочисленные команды и команды с плавающей точкой.

Таблица 1

Сравнение скоростей выполнения команд численной арифметики

Команда		Примерное отношение производительности: 66 МГц Pentium/66МГц Intel 486DX2
FADD	сложение	3,8
FDIV	деление	2,2
FYL2X	логарифм	3,1
FPATAN	арктангенс	2,6
F2XM1	экспонента	4,8
FLD	передача данных	4,0

2. Удобство использования

Непосредственный перевод математических численных алгоритмов в форму компьютерной программы может привести к появлению в ряде случаев неверных результатов вычислений. Гарантирование правильных результатов требует применения специальных сложных вычислительных методов, выходящих за пределы обычной практики программирования. Архитектура Intel спроектирована таким образом, что в большинстве случаев программист может создать надежную программу без использования этих специальных методов.

Например, рассмотрим процедуру умножения двух чисел с плавающей точкой с последующим делением результата на третье число. В большинстве компьютеров произойдет переполнение, даже если результат — корректное 32-разрядное число.

Другие характерные примеры: распространенное в финансовых вычислениях выражение $(1+i)^n$ или поиск корней квадратного уравнения

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Эта формула очень нестабильна в случае, если корни почти совпадают или, наоборот, очень сильно различаются. Вычисления по ней могут привести к ложным переполнениям или потере значимости при очень больших или очень маленьких значениях коэффициентов a, b, c.

Вычисления с помощью Pentium FPU дадут во всех этих случаях правильные, корректно округленные результаты, даже если числа, над которыми проводятся операции, заданы с обычной точностью (4 байта). Дело в том, что при вы-

числении подвыражений FPU автоматически выделяет для промежуточных результатов достаточно места, чтобы исключить угрозу переполнения или потери значимости.

Pentium FPU поддерживает точную арифметику для коммерческих вычислений, когда ошибки округления недопустимы, так как они приводят к денежным потерям. Точная арифметика допускает операции над целыми числами размерностью до 18 десятичных цифр (вплоть до 10^{18} или 2^{64}).

FPU также дает возможность обработки исключительных ситуаций (например, деление на 0 или $\sqrt{x}$ при $x<0$) без прекращения вычислений. По умолчанию исключительные ситуации обрабатываются с помощью специальных логических схем, входящих в состав FPU, возможно также подключение программного обработчика исключительных ситуаций.

3. Области применения

Использование возможностей Pentium FPU при реализации вычислительных алгоритмов целесообразно в следующих случаях:
- численные данные меняются в очень широком диапазоне или включают нецелые значения;
- алгоритмы порождают очень большие или очень маленькие промежуточные результаты;
- вычисления должны быть очень точными (должно быть сохранено большое количество значащих цифр);
- высокие требования к скорости выполнения вычислений;
- необходима гарантия надежности результатов, а программисты, реализующие алгоритмы, не являются специалистами по численным методам.

Приведем несколько характерных примеров приложений, в которых могут быть эффективно использованы возможности Pentium FPU.

1) Обработка коммерческих данных, в которых необходимы точные результаты операций над десятичными числами большой размерности. Финансовые вычисления, использующие возведение в степень.

2) Моделирование. Большое пространство памяти и высокая скорость процессора дают возможность решать задачи моделирования, для которых раньше гребовались дорогостоящие большие ЭВМ или мини-ЭВМ. Примеры: моделирование сложных электронных схем с использованием системы SPICE или моделирование больших механических систем с помощью метода конечных элементов.

3) Графические преобразования. FPU может использоваться в графических приложениях, требующих быстрых преобразований изображения, например, в САПР; это возможно при эффективном использовании возможности параллельной работы FPU и устройства для целочисленных операций.

4) Управление процессами в реальном времени, гребующее сочетания надежности вычислений с высокой скоростью.

5) Управление станками и роботами, требующее реализации алгоритмов перемещения и позиционирования инструментов в трехмерном пространстве (перемещение + вращение), выигрывает от аппаратной реализации тригонометрических функций и точности Pentium FPU.

6) Навигационные системы.

7) Обработка больших объемов данных.

8) Обработка цифровых сигналов. Характерные для используемых здесь алгоритмов операции умножения матриц и свертки могут эффективно использовать конвейерную организацию выполнения команд в Pentium.

4. Числовая система

Множество действительных чисел, рассматриваемое в формальной математике, бесконечно и непрерывно. Не существует верхнего или нижнего предела для величины числа или для его точности (количество значащих цифр, используемых в десятичном представлении числа). Между любыми двумя различными действительными числами находится бесконечное множество действительных чисел.

Конечность размеров регистров и оперативной памяти компьютера приводит к тому, что для моделирования системы действительных чисел используется конечное и дискретное множество чисел.

Диапазон нормализованных чисел с двойной точностью, которые могут выступать в Pentium FPU как окончательные результаты вычислений: $\pm 2{,}23 \times 10^{-308}$ — $\pm 1{,}79 \times 10^{308}$. Так как это множество чисел конечно, то при попадании результата вычислений в интервал между двумя смежными представимыми в FPU числами он подвергается округлению до одного из этих чисел, что приводит к потере точности. Представимые в FPU числа распределены по числовой оси неравномерно. Одно и то же количество представимых чисел находится между любыми двумя последовательными степенями двойки (то есть количество точно представимых в FPU чисел в интервале от 2 до 4 то же, что и в интервале от 65536 до 131072). Таким образом, пробелы между представимыми числами растут с ростом величины чисел. Точно представляются все целые в интервале от -2^{64} до 2^{64}.

В своих внутренних операциях для представления констант и промежуточных результатов вычислений Pentium FPU использует расширенную числовую систему с диапазоном представимых чисел от $\pm 3{,}37 \times 10^{-4932}$ до $\pm 1{,}18 \times 10^{4932}$.

С точки зрения практической используемая в Pentium числовая система достаточно велика и плотна для подавляющего большинства приложений.

5. Типы и форматы данных

Pentium распознает семь числовых типов данных для хранимых в памяти значений. Эти типы разделены на три класса: двоичные целые, упакованные десятичные целые и двоичные действительные. Форматы числовых типов данных представлены на рис. 1.

5.1. Двоичные целые

Три формата двоичных целых отличаются только длиной. Самый левый бит — знак числа (0 — положительное, 1 — отрицательное). Отрицательные числа представлены в дополнительном коде.

Все двоичные целые форматы существуют только в памяти. В FPU они автоматически преобразуются в 80-битовый расширенный действительный формат.

Формат данных	Диапазон	Точность	Наиболее значимый байт … Байт с большим адресом
Целое слово (word integer)	10^4	16 битов	15 … 0
Целое короткое (short integer)	10^9	32 бита	31 … 0
Целое длинное (long integer)	10^{18}	64 бита	63 … 0
Упакованное двоично-кодируемое десятичное (packed BCD)	10^{18}	18 цифр	S, X, d17, d16, d15 … d2, d1, d0; 79 … 72
Одинарная точность (single precision)	$10^{\pm 38}$	24 бита	S, порядок, МАНТИССА; 31, 23, 0
Двойная точность (double precision)	$10^{\pm 308}$	53 бита	S, порядок, МАНТИССА; 63, 52, 0
Расширенная точность (extended precision)	$10^{\pm 4932}$	64 бита	S, порядок, 1, МАНТИССА; 79, 63, 0

Рис. 1. Форматы числовых данных

5.2. Десятичные целые

Десятичные целые хранятся в упакованном виде, по две десятичные цифры в байте. Отрицательные числа отличаются от положительных только битом знака, который хранится в самом левом байте. Наиболее значимой является самая левая из цифр.

Десятичный целый формат существует только в памяти. В FPU он автоматически преобразуется в 80-битовый расширенный действительный формат.

5.3. Действительные числа

Pentium представляет действительные числа в форме $(-1)^S 2^E (b_0, b_1 b_2 \ldots b_{p-1})$.

Здесь S = 0 или 1 - знак,

E - любое целое между Emin и Emax,

b_i = 0 или 1,

p - точность.

Параметры трех форматов действительных чисел приведены в таблице 2.

Pentium хранит действительные числа в двоичном формате, соответствующем научной нотации записи чисел. Формат состоит из трех полей:

- мантисса $b_0, b_1 b_2 \ldots b_{p-1}$ состоит из значащих цифр числа;
- порядок (E + смещение) определяет место двоичной точки среди значащих цифр;
- 1-битовое поле знака (только оно отличает отрицательные числа от положительных).

FPU обычно хранит цифры мантиссы в нормализованной форме, то есть в виде $1, b_1 b_2 \ldots b_{p-1}$. Таким образом исключаются лидирующие нули маленьких чисел и уменьшается число значащих цифр. В форматах с одинарной и двойной точностью бит целой части мантиссы неявный и в действительности не хранится, физически он присутствует только в расширенном формате.

Для упрощения сравнения действительных чисел процессор хранит порядок в смещенной форме. Это значит, что к действительному значению порядка добавляется константа, называемая смещением и выбранная так, чтобы значение смещенного порядка было положительным. Это позволяет сравнивать два действительных числа одного и того же формата и знака как двоичные целые без знака, сравнивая их по битам слева направо.

Двоичные форматы с одинарной и двойной точностью существуют только в памяти; в FPU они автоматически преобразуются в расширенный формат.

Таблица 2

Параметры форматов действительных чисел

Параметр	Формат: Одинарная точность	Двойная точность	Расширенная точность
Ширина формата (битов)	32	64	80
p (число битов точности)	24	53	64
Число битов порядка	8	11	15
Emax	+127	+1023	+16383
Emin	-126	-1022	-16382
Смещение порядка	+127	+1023	+16383

В большинстве приложений следует использовать для хранения данных и результатов формат с двойной точностью; он в большинстве случаев обеспечивает диапазон и точность, достаточные для получения правильных результатов без дополнительных усилий со стороны программиста. Формат с одинарной точностью может быть использован, если задача требует жестких ограничений на занимаемую данными память, а также при отладке для быстрейшего выявления проблем, которые могут возникнуть из-за округления. Расширенный формат обычно резервируется лишь для хранения промежуточных результатов и констант.

5.4. Режимы округления

Если результат вычислений b является непредставимым в данном формате действительным числом, а два ближайших к нему представимых числа — a и c ($a < b < c$), то FPU может выбрать один из четырех следующих режимов округления.

1) Округление до ближайшего (режим, принятый по умолчанию). Выбирает из a и c число, ближе лежащее к b. Если расстояния до a и до c равны, то выбирается «четное» число (наименее значимый бит — 0).

2) Округление вниз (до a).

3) Округление вверх (до c).

4) Округление к нулю (до меньшего по абсолютной величине из a и c).

6. Архитектура устройства с плавающей точкой

С точки зрения программиста FPU состоит из набора регистров:

- 8 индивидуально адресуемых 80-битовых числовых регистров, образующих стек регистров;
- 16-битовое слово состояния FPU;
- 16-битовое управляющее слово FPU;
- слово меток регистров (или тэговое слово);
- два 16-битовых регистра, содержащих селекторы для последней команды и операнда;
- два 32-битовых регистра, содержащих смещения для последних команды и операнда;
- 11-битовый регистр, содержащий код последней неуправляющей команды FPU.

Все команды, выполняемые FPU, работают с содержимым этих регистров.

6.1. Стек регистров FPU

Все 8 числовых регистров, входящих в стек, имеют размер 80 битов и разделены на поля, соответствующие формату действительного числа с расширенной точностью. Стек регистров FPU изображен на рис. 2.

Команды FPU обращаются к регистрам стека, адресуясь относительно регистра-вершины стека. Номер регистра-вершины стека записан в поле TOP слова состояния FPU. Как и стек в памяти, стек регистров FPU растет сверху вниз по направлению к регистрам с меньшими номерами. Команды загрузки и занесения в стек уменьшают значение поля TOP на единицу и загружают значение в новый регистр-вершину стека. Команды записи в память и выталкивания сохраняют в памяти значение из текущего регистра-вершины стека и увеличивают поле TOP на 1.

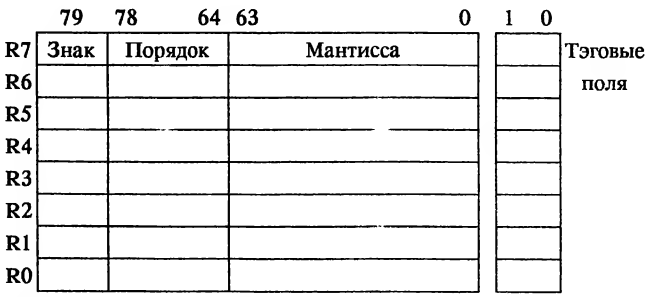


Рис. 2. Стек регистров FPU

Команды FPU либо неявно подразумевают операцию над регистром-вершиной стека, либо позволяют явно указывать положение регистров-операндов относительно вершины стека. Ассемблер ASM 386/486 поддерживает адресацию регистров с использованием выражения ST(0) или просто ST для регистра — вершины стека и ST(i) ($0 \le i \le 7$) для обращения к i-му от вершины регистру стека. Например, если регистр 3 — вершина стека (TOP содержит 011B), то сложение содержимого регистров 3 и 5 выполняется с помощью команды

```
FADD   ST, ST(2).
```

Стековая организация и адресация относительно вершины стека упрощают передачу параметров подпрограммам. До тех пор, пока стек неполон, каждая процедура перед вызовом подпрограммы загружает параметры в стек, а подпрограмма обращается к ним, как к ST, ST(1) и так далее, вне зависимости от того, какой регистр является вершиной стека.

Программист может работать с числовыми регистрами и как с набором разрозненных регистров: в этом случае ему следует использовать особенности конвейерной архитектуры Pentium и команду FXCH.

7. Набор команд FPU

Команды FPU в Pentium могут быть разделены на шесть функциональных групп.

1) Команды передачи данных (загрузка в регистр, запись в память, обмен содержимым регистров).

2) Нетрансцендентные команды (арифметические операции, извлечение корня, округление, получение абсолютного значения).

3) Команды сравнения.

4) Трансцендентные команды (тригонометрические функции, экспонента, логарифмирование).

5) Команды загрузки констант (помещают в стек часто используемые константы, такие как π, $\log_2 10$, $\log_e 2$).

6) Команды управления.

Полное описание команд FPU содержится в [1].

8. Различия реализаций системы команд с плавающей точкой в разных архитектурах Intel

Система команд с плавающей точкой математического сопроцессора Intel 387, и FPU Intel 486 и Pentium содержит ряд команд, отсутствующих в системе команд сопроцессоров 8087 и Intel 287; характер выпол-

нения некоторых команд изменился. Остановимся подробнее на различиях в командах с плавающей точкой для разных архитектур Intel и возможных изменениях в программах при переходе от одной архитектуры к другой.

Говоря об архитектурах Intel 387 NPX, Intel 486 и Pentium, мы будем употреблять слово «теперь», а Intel 287 и 8087 NPX — «раньше».

1) FDIV, FPREM, FSQRT — теперь эти операции поддерживаются над ненормализованными операндами и может случиться исключительная ситуация «потеря значимости». Раньше такие операции вызывали исключительную ситуацию «неверная операция» (invalid operation), и потеря значимости была невозможна. Может потребоваться внесение изменений в обработчик исключительных ситуаций для потери значимости, если он по-разному обрабатывал разные коды операций. Причина изменений — обеспечение совместимости со стандартом IEEE 754.

2) FSCALE — диапазон масштабирующего операнда теперь не ограничен. При $0 < |ST(1)| < 1$ ST(0) не меняется. Раньше этот диапазон был ограничен, а при $0 < |ST(1)| < 1$ результат не был определен. Воздействие на программное обеспечение: при $0 < |ST(1)| < 1$ будут выдаваться разные результаты. Изменение произведено для расширения диапазона масштабирования.

3) FPREM1 — вычисляет остаток от деления по правилам стандарта IEEE 754. Раньше этой команды не было, введена для совместимости со стандартом.

4) FPREM — теперь биты C0, C3 и C1 слова состояния точно отражают содержимое трех младших битов частного. Раньше эти биты были неверны при делимом вида 64^N+M, если $N \geq 1$, а M=1 или M=2. На программы это изменение не влияет.

5) FUCOM, FUCOMP, FUCOMPP выполняют неупорядоченное сравнение в соответствии со стандартом IEEE 754. Раньше этих команд не было, введены для совместимости со стандартом.

6) FPTAN — диапазон операнда теперь расширен ($|ST(0)| < 2^{63}$). Раньше диапазон был ограничен ($|ST(0)| < \pi/4$), и операнд должен был сводиться к этому диапазону при помощи команды FPREM; теперь это происходит с большей точностью с помощью внутренней константы $\pi/4$. Воздействия на программы нет, за исключением возросшей производительности.

7) Теперь по-новому, в соответствии со стандартом IEEE 754, обрабатывается маскируемое переполнение стека. На программы это не влияет.

8) FSIN, FCOS, FSINCOS — этих команд, реализующих обычные тригонометрические функции, раньше не было. Их появление не влияет на существующие программы, но позволяет повысить производительность.

9) FPATAN — теперь диапазон операндов не ограничен; раньше $|ST(0)|$ должен был быть меньше, чем $|ST(1)|$. На совместимость программ не влияет; ведет к росту производительности.

10) F2XM1 — теперь диапазон операндов $-1 < ST(0) < +1$; раньше $0 \leq ST(0) \leq 0{,}5$.

11) FLD extended-real — теперь не сообщается об исключительной ситуации, так как инструкция не арифметическая; раньше сообщалось. Не влияет на совместимость.

12) FXTRACT — теперь если операнд — 0, сообщается об исключительной ситуации «деление на 0», в ST(1) помещается $-\infty$. Если операнд $+\infty$, исключительной ситуации нет. Раньше, если операнд равнялся 0, в ST(1) помещался 0 и не было исключительной ситуации, а если операнд был $+\infty$, то сообщалось об исключительной ситуации «неверная операция». Влияния на программы нет, так как программы обычно пренебрегают значениями 0 и $+\infty$. Изменения соответствуют рекомендациям IEEE 754.

13) FLD constant — теперь работает управление округлением, которого не было раньше. Отличается наименее значимый бит мантиссы результата, если управление установлено на округление до $-\infty$ или округление до 0 в FLDPI, FLDLN2, FLDLG2, FLDL2E; для FLDL2T — если задано округление до $+\infty$. Это соответствует IEEE 754.

14) FLD single/double precision — теперь загрузка денормализованного числа влечет его преобразование в формат с расширенной точностью; раньше оно преобразовывалось по-другому. Если следующая команда — FXTRACT или FXAM, результат изменяется. Изменение сделано для совместимости с IEEE 754.

15) FLD single/double precision — при загрузке сигнального NaN теперь возникает исключительная ситуация («invalid»). Раньше ее не было. Должен быть изменен обработчик исключительных ситуаций. Изменение сделано для совместимости с IEEE 754.

16) FSETPM — теперь эта команда трактуется как FNOP. Раньше она сообщала сопроцессору Intel 287, что система в защищенном режиме. Влияния на программы нет.

17) FXAM — теперь, если встретился пустой регистр, комбинации C3-C0, равные 1101 или 1111 не генерируются. Раньше они были возможны. На программы это не влияет, но увеличивается производительность.

18) FSAVE, FSTENV — теперь адрес указателя на операнд в памяти, в который происходит запись, не определен, если предыдущая ESC — команда не ссылается на память.

Полное и точное описание отличий в реализации системы команд с плавающей точкой в разных архитектурах Intel см. [1].

Автор выражает благодарность сотрудникам московского представительства Intel Corporation за содействие при подготовке статьи. ■

Литература

1. **Pentium Processor User's Manual.– USA: Intel Corporation, 1993.– Vol. 3: Architecture and Programming Manual.**

Редакция журналов «Монитор» и «Монитор-Аспект» приглашает к сотрудничеству специалистов, желающих принять участие в работах по тестированию программных продуктов и аппаратуры.

Обращаться по тел. 335-68-31.
Дмитрий Бройтман

У Вас накопилась такая большая картотека, что работа с ней превратилась в сплошные мучения?
Папки с документами, вырезками и текстами заполонили полки и найти что-либо нужное уже не представляется возможным?

Всем пользователям IBM PC ИНФОРКОМ предлагает DB–PROCESSOR Версия 4.1 — лучшее средство от головной боли

Вы хотите иметь одну или несколько специализированных баз данных, отвечающих Вашим специфическим потребностям?
Вам не нужно обращаться к разработчикам — при помощи DB-PROCESSOR Вы сможете самостоятельно без особого труда создать неограниченное число разнообразных баз данных.

Вы не знаете ни языков программирования, ни даже английского языка?
Не беда — при работе с DB-PROCESSOR Вам не потребуется ни то ни другое.

Вы боитесь, что при работе с компьтером можете сделать что-то не так?
Ваши страхи напрасны: DB-PROCESSOR обеспечивает 100% защиту от возможных некорректных действий пользователя.

Вам необходимо хранить большое количество текстового материала (договоры, истории болезней, цитаты и так далее), быстро находить нужные тексты, вносить в них изменения и распечатывать на принтере?
DB-PROCESSOR обеспечит Вам все эти возможности.

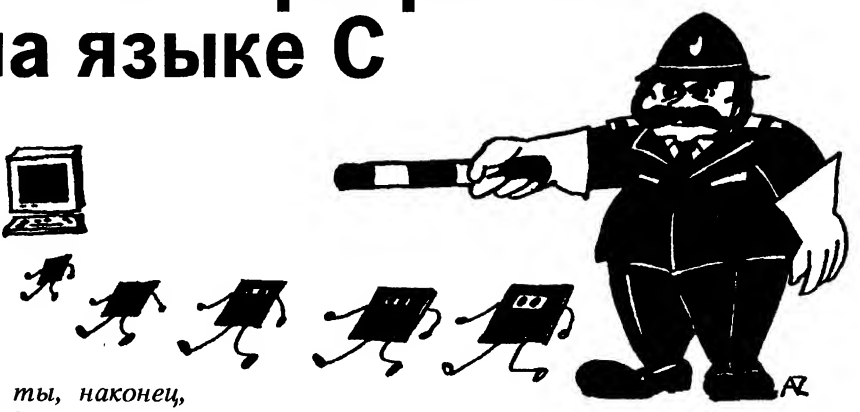
В Вашей фирме установлена локальная сеть?
DB-PROCESSOR чувствует себя в локальной сети как рыба в воде.

Мы на своем опыте убедились: работа с DB–PROCESSOR вызывает только положительные эмоции.

Звоните: (095) 452-31-03 (вторник, среда, четверг).
Пишите: 121019 Москва, Г-19, а/я 16, ИНФОРКОМ

Резидентные программы на языке С

Д.Валдин



Век живи — век учись! И ты, наконец, достигнешь того, что, подобно мудрецу, будешь иметь право сказать, что ничего не знаешь.

Козьма Прутков

Часть 1

С момента создания фирмой Microsoft первой версии операционной системы для IBM PC прошло уже более десяти лет. И все эти годы многочисленная армия программистов упорно не желала мириться с тем фактом, что MS-DOS является по своей природе однозадачной системой. Первым доказательством тому послужил выпуск фирмой Rosesoft в 1982 году коммерческой резидентной (TSR — Terminate and Stay Resident) программы PROKEY, работающей под управлением MS-DOS. Фирма Borland не пожелала остаться в тени, и вскоре на рынке программного обеспечения появился небезызвестный Sidekick, который по уровню продаж превзошел все ожидания его создателей. На сегодняшний день практически каждая фирма, специализирующаяся на разработке программного обеспечения, имеет за плечами опыт создания TSR программ. Тем не менее эта задача по-прежнему остается одной из наиболее трудных задач программирования. Отважившемуся на интервенцию в страну DOS необходимы не только знания особенностей этой системы, но и знакомство с аппаратной частью.

Чем сложнее задача, тем интереснее ее решать и тем сильнее радость, которую испытываешь от сознания того, что еще один барьер незнания остался позади. Попробуем и мы, руководствуясь известным изречением Козьмы Пруткова, постичь премудрости создания TSR программ. А поможет нам в этом язык программирования С.

Наверное, я не ошибусь, предположив, что в Вашем распоряжении имеется один из компиляторов языка С фирм Microsoft или Borland. Если это действительно так, у нас с Вами не будет никаких проблем. Если же Вы предпочитаете продукцию других фирм или, что более вероятно, другой язык программирования, не огорчайтесь. Возможно, Вы сможете найти в этой статье что-то полезное и для себя. Во всяком случае, мне бы очень хотелось, чтобы это было именно так.

«Начнем, пожалуй.»

Как оставить программу резидентной

Первый вопрос, который нам предстоит решить, — как оставить программу резидентной. Воспользуемся для этого библиотечной функцией _dos_keep()*. Она использует обращение к функции 31h DOS и имеет прототип:

```
void _dos_keep(unsigned retcode
               unsigned memsize );
```

Функция завершает вызвавший ее процесс, оставляя его резидентным в памяти. Затем она возвращает младший байт retcode (код завершения) процессу, являющемуся родительским по отношению к резидентному (в простейшем случае командному интерпретатору COMMAND.COM). Размер памяти, занимаемой резидентным процессом, устанавливается равным memsize параграфов (1 параграф = 16 байтам). Излишки памяти возвращаются операционной системе.

С первым параметром все понятно: его значение будет присвоено глобальной переменной ERRORLEVEL при завершении процесса. Поэтому мы вправе присваивать ему любое значение. А вот со вторым возникают определенные трудности. Как определить размер памяти, необходимый для нашей TSR программы? Не будем

* Все обслуживающие функции DOS библиотеки Microsoft имеют вид _dos_XXX(). У их аналогов из библиотеки Borland префикс _dos_ отсутствует.

убеждать себя в том, что он равен размеру EXE файла. Ведь помимо кода программы и глобальных данных где-то в памяти должны находиться области стека и динамических данных. Поэтому для определения memsize необходимо представлять себе, как код программы и данные размещаются в памяти. Поскольку фирмы Microsoft и Borland по-разному решают эту задачу, приведем карты распределения памяти (рис. 1) для программ, построенных компиляторами обеих фирм (Small model).

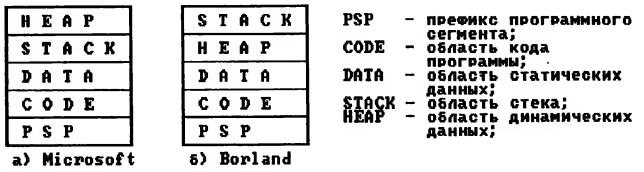


Рис. 1. Карты распределения памяти для компиляторов фирм Microsoft и Borland

С учетом того, что стек программы увеличивается по направлению к области младших адресов, размер памяти, необходимый для нашей TSR программы, можно определить как разницу между вершиной стека и началом PSP. Причем в случае использования компиляторов фирмы Microsoft к полученному результату необходимо прибавить размер области динамических данных. Однако возникает новый вопрос: как определить размер области динамических данных? Мы знаем, что память для хранения динамических объектов выделяется посредством обращения к функциям **malloc()**, **calloc()** и так далее. Вопрос кажется почти решенным: достаточно лишь выяснить, какой максимальный объем динамической памяти требуется нашей программе, — и дело сделано. Однако не будем забывать, что многие библиотечные функции также обращаются к **malloc()**. Это функции буферизованного ввода-вывода (**getchar()**, **putchar()**, **printf()**), которые резервируют память для буферизации данных. Это также функции **fopen()**, **strdup()** и некоторые другие. Более подробную информацию о них можно получить в литературе.

Итак, будем считать, что размер области динамических данных определен. Посмотрим еще раз на карту распределения памяти программы, построенной компилятором фирмы Borland. Область динамических данных располагается между областями статических данных и стека. По умолчанию размер этой области устанавливается максимально возможным. Однако существует возможность изменения этого значения. Для этого достаточно переопределить значение глобальной переменной **_heaplen**, например,

```
unsigned _heaplen = 1024; // байта.
```

Размер стека для программ, написанных на языке С, по умолчанию устанавливается равным 2 Кбайтам для компиляторов фирмы Microsoft и 4 Кбайтам для Borland. В TSR программах зачастую можно обойтись стеком значительно меньшего размера. Если программа будет компилироваться в среде Borland, то для изменения глубины стека, установленной по умолчанию, следует присвоить новое значение глобальной переменной _stklen, например:

```
unsigned _stklen = 512;     // байтов
```

Компиляторы Microsoft не предоставляют такой возможности. Размер стека указывается на этапе компоновки. Поэтому при их использовании необходимо специфицировать размер стека в командной строке, например:

```
cl hello.c /link /ST:0x200
```

или установить его в подменю <Linker Flags>.

На примере простейшей резидентной программы покажем алгоритмы определения ее размера, ориентированные на использование компиляторов фирм Microsoft и Borland (листинги 1 и 2).

Некоторые TSR программы перед тем как остаться резидентными освобождают выделенный им при старте блок памяти, содержащий копию среды DOS. Сделать это нетрудно. Сегментный адрес этого блока находится в слове, расположенном со смещением +2CH относительно начала PSP:

```
_dos_freemem( peek( _psp, 0x2c ));
```

Однако вряд ли игра стоит свеч. Большого выигрыша мы не добьемся, зато лишим себя удовольствия лицезреть имя своей бессмертной программы в списке резидентных процессов, выдаваемом различными детекторами.

Закончив ввод программы, не будем торопиться сразу же запускать ее и на компиляцию, и на исполнение. Вспомним, что TSR программы нельзя запускать, находясь в интегрированной среде, поскольку это может вызвать серьезные проблемы, связанные с распределением памяти. Предположим, что мы сделали все как надо. Давайте убедимся, что исполненная нами программа действительно осталась резидентной в памяти. Воспользуемся для этого утилитой MEMMAP или аналогичной. Мы видим, что размер нашей TSR программы лишь незначительно превышает размер EXE файла. Значит, все сделано правильно, и нам пора переходить ко второму вопросу: каким образом наша резидентная программа может получить управление. Чтобы получить ответ на этот вопрос, освежим наши знания о системе прерываний. Обратимся к классикам [1–4].

Система прерываний

Когда программе или периферийному устройству требуются функции обслуживания операционной системы или BIOS, они могут прервать работу процессора (текущую программу), чтобы передать управление нужной системной программе, которая после окончания работы вернет управление в точку прерывания.

Существуют два класса прерываний: внутренние и внешние.

Внутренние прерывания инициируются состоянием центрального процессора или специальной командой. К ним относятся:

- прерывание при делении на ноль;
- прерывание по флагу TF (Trace Flag — флаг трассировки); используется программами отладки типа DEBUG;
- инструкции INT (INTerrupt — прерывание) и INTO (INTerrupt Overflow — прерывание при переполнении); эти прерывания называются

также программными; в качестве операнда инструкции INT задается номер прерывания, которое нужно выполнить, например, INT 10H — видеосервис, INT 16H — сервис клавиатуры.

Программные прерывания - это особая форма обращения к процедуре, при которой вызывающая программа может не знать адреса вызываемой процедуры, а знать лишь номер прерывания, соответствующего ей. Оригинальные процедуры программных прерываний (Interrupt Handlers — обработчики прерываний) являются принадлежностью операционной системы. Некоторые из них, например, INT 9H, INT 10H, INT 16H находятся в ПЗУ (в BIOS); другие: INT 20H, INT 21H, INT 25H/26H, — в DOS. Все эти процедуры предназначены для выполнения таких рутинных работ, как обработка нажатия клавиши на клавиатуре, вывод на экран, запись на диск и так далее. Механизм прерываний дает возможность обращаться к этим процедурам из любой программы. Поэтому программные прерывания можно рассматривать как удобный способ взаимодействия прикладных программ с операционной системой.

Programmer

С-компиляторы фирм Microsoft и Borland имеют различные соглашения о последовательности сохранения регистров в стеке.

При программировании на языках высокого уровня весь интерфейс с операционной системой реализуется на уровне библиотечных функций, поэтому к непосредственному вызову программных прерываний приходится прибегать лишь в исключительных случаях, связанных, как правило, с использованием недокументированных возможностей DOS. Однако применение библиотечных функций в TSR программах допустимо только тогда, когда программист четко представляет себе механизм их работы. В противном случае неизбежны ошибки, приводящие к нарушению работы операционной системы и, возможно, зависанию компьютера.

Программные прерывания могут вызываться друг из друга. Например, все функции обработки ввода с клавиатуры DOS (INT 21H) используют прерывание обработки ввода с клавиатуры BIOS (INT 16H). Аппаратные прерывания также могут прибегать к услугам программных. Так, например, прерывание от системного таймера INT 8H вызывает программное прерывание INT 1CH.

Процедуры программных прерываний, как правило, не связываются с конкретной программой, а размещаются в фиксированных областях памяти. Поэтому они могут использовать для взаимодействия с другими программами только общие области с абсолютной адресацией.

Передача параметров обработчику прерывания осуществляется через регистры, использование которых зависит от номера прерывания. Некоторые процедуры прерываний предназначены для выполнения не одной, а нескольких функций, например, INT 10H, INT 13H, INT 16H, INT 21H. В этом случае номер функции, которую необходимо выполнить, задается в регистре AH. Остальные регистры загружаются в зависимости от номера функции. Источники, содержащие информацию об использовании обслуживающих функций DOS и BIOS, весьма разнообразны. В частности, можно порекомендовать On-line HELP, поставляемый вместе с компиляторами фирмы Microsoft.

Внешние прерывания возникают, когда устройство ввода-вывода требует обслуживания центрального процессора. Существуют два сигнала, подачей которых на одноименные выводы процессора можно прервать его работу:

- INTR (INTerrupt Request — заявка на прерывание);
- NMI (Non-Maskable Interrupt — немаскируемое прерывание).

Соответственно, внешние прерывания подразделяются на маскируемые и немаскируемые.

Маскируемые прерывания, их называют также **аппаратными прерываниями**, генерируются контроллером прерываний Intel 8259A по заявке определенных периферийных устройств. Контроллер прерываний имеет 8 уровней приоритета IRQ0 — IRQ7. В IBM PC AT установлены два контроллера, что позволило увеличить количество уровней приоритетов периферийных устройств до 16. При одновременном поступлении нескольких запросов на прерывание в первую очередь обслуживается устройство с более высоким приоритетом.

Уровни приоритетов определены аппаратно. Наивысший приоритет (IRQ0) предоставлен системному таймеру (INT 8H). Такой выбор не случаен. Ведь если прерывание от него будет все время теряться, то неверными окажутся показания системных часов. На втором месте — клавиатура (IRQ1, INT 9H). Прерывания от системного таймера и клавиатуры являются теми двумя аппаратными прерываниями, которые чаще всего перехватываются TSR программами.

Существует возможность программного назначения низкого приоритета любому из уровней IRQ с циклическим изменением остальных приоритетов. Если низкий приоритет назначить IRQ4, получится следующий порядок приоритетов:

IRQ5, IRQ6, IRQ7, IRQ0, IRQ1, IRQ2, IRQ3, IRQ4.

Однако вряд ли подобный «трюк» с реорганизацией приоритетов представляет для нас практический интерес.

Аппаратные прерывания могут быть запрещены программно, причем необходимость запрета обуславливается двумя причинами.

В первом случае блокируются все аппаратные прерывания, с тем чтобы критическая часть кода была выполнена целиком, прежде чем компьютер произведет какое-либо другое действие. Например, аппаратные прерывания запрещают при переключении стека или при изменении вектора аппаратного прерывания, чтобы избежать выполнения прерывания, когда вектор изменен только наполовину.

Во втором случае маскируются только определенные аппаратные прерывания. Это делается, когда некоторые прерывания могут взаимодействовать с операциями, критичными к временным интервалам. Например, точно рассчитанная по времени процедура ввода-вывода не может позволить себе быть прерванной длительным дисковым прерыванием.

Для запрета всех аппаратных прерываний используется инструкция cli, которая сбрасывает флаг IF (Interrupt Flag — флаг разрешения маскируемых прерываний). Если запрос на прерывание приходит в момент, когда аппаратные прерывания запрещены, он запоминается и начинает обслуживаться только тогда, когда будут разрешены прерывания. Все последующие запросы будут игнорироваться контроллером прерываний до тех пор, пока не будет обработано задержанное прерывание. По этой причине допускается запрещать прерывания лишь на очень короткие промежутки времени, необходимые для исполнения критического участка программы. Для разрешения аппаратных прерываний служит инструкция sti, которая устанавливает флаг IF. Повторное выполнение команд cli, если IF уже сброшен, и sti, если он установлен, не оказывает никакого влияния на микропроцессор. И последнее, на что хотелось бы обратить внимание. Флаг IF никак не влияет на выполнение программных прерываний. Мало того, при каждом вызове программного прерывания микропроцессор автоматически сбрасывает этот флаг, запрещая тем самым аппаратные прерывания. Поэтому, если процедура программного прерывания требует значительных временных затрат, то в самом ее начале следует поместить инструкцию sti, разрешив таким образом аппаратные прерывания.

Запрет отдельных аппаратных прерываний осуществляется записью единицы в соответствующий разряд IMR (Interrupt Mask Register — регистр маски прерывания) контроллера Intel 8259. Для разрешения прерывания в тот же разряд необходимо записать ноль. Адреса связанных с этим регистром портов ввода-вывода и номера битов маски приведены в таблице 1.

Логика контроллера 8259A предусматривает возможность его работы в нескольких режимах. В режиме обычных приоритетов при установленном флаге IF запросы с приоритетами выше обрабатываемого вызывают прерывание текущей процедуры прерывания, а запросы с меньшими приоритетами ожидают своей очереди.

Каждому запросу соответствует свой бит в ISR (Interrupt Service Register — регистр обслуживания прерывания) — регистре, предназначенном для хранения текущих обрабатываемых запросов. При установленном бите ISRn шифратор приоритетов не распознает запросов на линиях IR(n+1) — IR7, но распознает незамаскированные запросы на входах IR0 — IR(n-1). Таким образом, процедура прерывания с меньшим приоритетом может прервать выполнение процедуры с большим приоритетом только в случае, если та сбросит соответствующий ей бит в ISR.

Существует, однако, возможность заставить контроллер игнорировать схему приоритетов и обрабатывать незамаскированные запросы по мере их поступления (если, конечно, IF = 1).

При написании обработчика аппаратного прерывания программист должен позаботиться о том, чтобы процедура прерывания сбрасывала соответствующий ей бит в ISR. Сделать это можно либо передав управление оригинальному обработчику прерывания, либо послав сигнал EOI (End Of Interrupt — конец прерывания) контроллеру прерывания:

```
outp( 0x20, 0x20 );
```

Немаскируемое прерывание инициируется сигналом, подаваемым на вход NMI. Оно имеет номер 02H, обладает самым высоким приоритетом и вызывается независимо от состояния флага IF. Сигнал немаскируемого прерывания обычно формируют схемы, фиксирующие такие события, как сбои питания, ошибки в канале ввода-вывода и некоторые ошибки сопроцессора. Заметим, что никаких ограничений на использование этого прерывания в прикладных программах не существует.

Таблица 1

Аппаратные прерывания IBM PC

IRQ	Номер прерывания	Адрес порта	Номер бита регистра маски	Устройство
0	08H	21H	0	системный таймер
1	09H	21H	1	клавиатура
2	0AH	21H	2	канал ввода-вывода
8	70H	A1H	0	часы реального времени (AT)
9	71H	A1H	1	программно переводим в IRQ2
10	72H	A1H	2	резерв
11	73H	A1H	3	резерв
12	74H	A1H	4	резерв
13	75H	A1H	5	сопроцессор (AT)
14	76H	A1H	6	жесткий диск (AT)
15	77H	A1H	7	резерв
3	0BH	21H	3	COM1 (COM2 на AT)
4	0CH	21H	4	COM2 (COM1 на AT)
5	0DH	21H	5	жесткий диск на XT (LPT2 на AT)
6	0EH	21H	6	контроллер НГМД
7	0FH		7	LPT1

Независимо от вида прерывания возникающие при этом действия одинаковы и называются последовательностью прерывания. До того как процессор может приступить к выполнению этой последовательности, он должен завершить выполнение текущей команды, если ею не являются команды HLT или WAIT. После распознавания прерывания процессор выполняет следующие действия:

1. Определяет N — номер прерывания.
2. Помещает в стек текущее содержимое PSW (Program Status Word — слово состояния процессора), CS (сегмент кода) и IP (указатель команды).
3. Сбрасывает флаги IF и TF, запретив тем самым маскируемые прерывания и прерывания по флагу трассировки.
4. Загружает регистр IP содержимым ячейки памяти с абсолютным адресом 4*N, а регистр CS — содержимым 4*N+2.

Таким образом управление передается процедуре прерывания, по завершении которой оно возвращается прерванной программе. Процедуры аппаратных прерываний заканчиваются инструкцией IRET (Interrupt RETurn — возврат из прерывания), которая восстанавливает из стека сохраненные там значения PSW, CS, IP. Однако некоторые процедуры программных прерываний заканчиваются инструкцией RET+2, что позволяет использовать регистр флагов как индикатор ошибки. Например, большинство функций DOS (INT 21H) устанавливают CF (Carry Flag — флаг переноса) в случаях неуспешного завершения процедуры прерывания.

Двойное слово, в котором находится новое содержимое IP и CS, называется вектором прерывания. Векторы прерываний образуют таблицу, расположенную в самых младших адресах оперативной памяти. Таблица содержит 256 векторов и занимает 1 Кбайт. Каждый вектор прерывания имеет свой номер, называемый номером прерывания, который указывает его место в таблице. Так, вектор прерывания от клавиатуры (INT 9H) размещается в байтах 00024...00027. При запуске системы векторы инициализируются таким образом, чтобы они указывали на программы, являющиеся частью DOS, BIOS, либо на системные таблицы. Пользовательским программам разрешается изменять эти векторы по своему усмотрению, с тем чтобы они указывали на новые процедуры прерываний. Этот процесс называется перехватом прерывания.

Стоп! Ведь это как раз то, что нам нужно. Напишем собственный обработчик прерывания от системного таймера и включим его в текст нашей TSR программы. Теперь перед обращением к функции _dos_keep() изменим вектор прерывания INT 8H так, чтобы он указывал на наш обработчик. После того как программа будет оставлена резидентной в памяти, этот обработчик будет получать управление всякий раз, когда будет генерироваться прерывание от системного таймера.

С тем же успехом мы могли бы перехватить программное прерывание INT 1CH (напомним, что оно вызывается из INT 8H), оригинальный обработчик которого состоит из одной-единственной инструкции IRET.

Если же мы захотим, чтобы наша TSR программа активизировалась при нажатии определенной «горячей клавиши» (Hot Key), как, например, THELP, Norton Guide, то нам потребуется перехватить аппаратное прерывание от клавиатуры (INT 9h) или программное прерывание сервиса клавиатуры (INT 16H).

Средства языка С для работы с прерываниями

Для чтения или изменения вектора прерывания можно использовать операции прямого доступа к памяти, однако будет лучше, если мы последуем рекомендациям, приведенным в руководстве по операционной системе MS-DOS, и воспользуемся функциями 25H/26H DOS. Стандартная библиотека языка С содержит функции **_dos_getvect()** и **_dos_setvect()**, которые реализуют вызовы этих обслуживающих функций DOS. Они имеют прототипы:

```
void (   _interrupt _far
       *_dos_getvect( unsigned intnum ))();
void _dos_setvect(unsigned intnum,
       void (_interrupt _far *handler)() );
```

Расширение стандарта ANSI, поддерживаемое компиляторами языка С фирм Microsoft и Borland, допускает использование ключевого слова **_interrupt** в качестве модификатора функции. Встретив такой модификатор, компилятор трактует функцию как обработчик прерывания и формирует соответствующую последовательность инструкций, включающую в себя:

- сохранение регистров;
- инициализацию регистра DS значением сегмента данных С программы;
- инициализацию регистра BP значением SP;
- сброс флага направления (только для компиляторов фирмы Microsoft);
- восстановление регистров;
- завершение функции инструкцией IRET — возврат из прерывания.

К сожалению, фирмы Microsoft и Borland имеют различные соглашения о последовательности сохранения регистров в стеке, что приходится учитывать при разработке программ, рассчитанных на компиляторы обеих фирм. В примерах 1 и 2 приведен код, полученный при компиляции функции **newint()** компиляторами Microsoft C 6.0 и Borland C 2.0.

Функция **newint()** может быть декларирована с параметрами. Мы приведем лишь вариант (пример 3), ориентированный на компиляторы фирмы Microsoft (Microsoft C 6.0, Quick C 2.5). Однако аналогичные рассуждения справедливы и для компиляторов фирмы Borland (Turbo C++ 1.0, Borland C 2.0).

В этом случае появляется возможность доступа к значениям всех регистров, какими они были в точке прерывания. Формальные параметры (псевдорегистры) перечисляются в порядке, обратном тому, в котором они помещаются в стек. Последние три параметра (_ip, _cs, _flags) содержат значения регистра указателя команды IP, сегментного регистра кода CS и слова состояния процессора PSW. Мы помним, что значения PSW, CS, IP (именно в таком порядке) помещаются в стек микропроцессором сразу же после того, как им будет определен тип прерывания.

Параметры, находящиеся в конце списка, разрешается опускать, если нет необходимости доступа к ним

Пример 1. Код функции _interrupt, генерируемый компиляторами Microsoft

```
void _interrupt _far newint( void )
{
}

; Compiler: Microsoft C 6.0
_newint     PROC    FAR
            push    ax
            push    cx
            push    dx
            push    bx
            push    sp
            push    bp
            push    si
            push    di
            push    ds
            push    es
            mov     bp,sp
            mov     ax,DGROUP
            mov     ds,ax
            cld
;
            pop     es
            pop     ds
            pop     di
            pop     si
            pop     bp
            pop     bx
            pop     bx
            pop     dx
            pop     cx
            pop     ax
            iret

_newint     ENDP
```

Пример 2. Код функции _interrupt, генерируемый компилятором фирмы Borland

```
; Compiler: Borland C++ 2.0
_new_int    proc    far
            push    ax
            push    bx
            push    cx
            push    dx
            push    es
            push    ds
            push    si
            push    di
            push    bp
            mov     bp,DGROUP
            mov     ds,bp
            mov     bp,sp
;
            pop     bp
            pop     di
            pop     si
            pop     ds
            pop     es
            pop     dx
            pop     cx
            pop     bx
            pop     ax
            iret
_new_int    endp
```

Пример. 3. Определение функции обработки прерывания с формальными параметрами

```
void _interrupt _far newvideo( unsigned _es, unsigned _ds,
                               unsigned _di, unsigned _si,
                               unsigned _bp, unsigned _sp,
                               unsigned _bx, unsigned _dx,
                               unsigned _cx, unsigned _ax,
                               unsigned _ip, unsigned _cs,
                               unsigned _flags );
```

из функции обработчика прерывания. Например, если обработчику требуется значение регистра DI, то достаточно указать только первые три параметра (_es, _ds, _di). Важно помнить, что код, генерируемый компилятором, не зависит ни от числа формальных параметров, ни от их имен. Таким образом, какие бы имена ни были указаны в списке формальных параметров, первый всегда будет содержать значение регистра ES, второй — регистра DS и так далее. Вполне естественно, что ошибиться при перечислении параметров не составляет никакого труда, особенно если в программе присутствует несколько обработчиков прерываний и каждый со своим списком параметров. Поэтому представляется разумным определить макрос ARGS (пример 4), и все обработчики прерываний, использующие значения регистров, объявлять как:

```
void _interrupt _far newint( ARGS );
```

Пример 4. Макроопределение для облегчения формирования списка формальных параметров

```
#define ARGS    unsigned _es, unsigned _ds, unsigned _di, \
                unsigned _si, unsigned _bp, unsigned _sp, \
                unsigned _bx, unsigned _dx, unsigned _cx, \
                unsigned _ax, unsigned _ip, unsigned _cs, \
                unsigned _flags
```

Если вспомнить о таком мощном средстве языка C, как условная компиляция, то с учетом особенностей конкретного компилятора можно объявить макрос ARGS, как показано в примере 5.

Пример 5. Макроопределение формальных аргументов для компиляторов Microsoft и Borland

```
#if defined _MSC_VER
#define ARGS    unsigned __es, unsigned __ds, unsigned _di, \
                unsigned _si, unsigned _bp, unsigned _sp,   \
                unsigned _bx, unsigned _dx, unsigned _cx,   \
                unsigned _ax, unsigned _ip, unsigned __cs,  \
                unsigned _flags
#elif defined __TURBOC__
#define ARGS    unsigned _bp, unsigned _di, unsigned _si,   \
                unsigned __ds, unsigned __es, unsigned _dx, \
                unsigned _cx, unsigned _bx, unsigned _ax,   \
                unsigned _ip, unsigned __cs, unsigned _flags
#endif
```

Формальные параметры, соответствующие сегментным регистрам CS, ES и DS, объявлены с двумя символами подчеркивания, поскольку _cs, _es и _ds являются для компиляторов фирмы Borland зарезервированными словами.

Взаимодействие со старым обработчиком прерывания

Практически в 100% случаев реализуемый нами обработчик прерывания, во избежание конфликтов между программами, должен взаимодействовать со старым обработчиком. Руководство по пакету Microsoft C 6.0 предлагает следующие два способа такого взаимодействия.

1. Передача управления старому обработчику посредством функции _chain_intr()

Функция **_chain_intr()** (ее аналог отсутствует в пакетах фирмы Borland) является, по существу, программным трюком. Функция работает корректно только при условии, что она вызвана из обработчика прерывания, написанного на C, то есть функции, декларированной с ключевым словом **_interrupt**. Получив в качестве аргумента адрес старого обработчика прерывания, функция **_chain_intr()** восстанавливает из стека значения всех регистров (в том числе и регистра DS) и, манипулируя со стеком, передает управление старому обработчику таким образом, что по окончании его работы возврат осуществляется не на оператор, следующий за вызовом функции **_chain_intr()**, а непосредственно в точку прерывания (пример 6).

Переменная oldint должна быть объявлена следующим образом:

```
void (_interrupt _far *oldint)();
```

Хотя это и напоминает, скорее, прототип функции, тем не менее именно так следует объявлять перемен-

Пример 6. Вызов функции _chain_intr()

```
void _interrupt _far newint()
{
  ...
  _chain_intr( oldint );

  /* Этот код никогда не выполняется. */
  ...
}
```

ную, являющуюся указателем на функцию-обработчик прерывания.

Очевидно, что переменная oldint должна получить значение старого вектора прерывания прежде, чем будет установлен новый обработчик прерывания:

```
oldint = _dos_getvect( intno );
```

где intno — номер вектора прерывания.

Если требуется передать старому обработчику измененное значение какого-либо регистра, то перед вызовом функции **_chain_intr()** необходимо присвоить это значение формальному параметру, соответствующему этому регистру. Так, если мы хотим, чтобы при передаче управления старому обработчику в регистре DI находилась единица, то мы должны включить в программу следующий оператор:

```
_di = 1;
```

При этом сохраненное в стеке значение регистра DI будет изменено на единицу и именно эта единица будет загружена в регистр DI после того, как функция **_chain_intr()** восстановит из стека сохраненные там значения всех регистров. Заметим, что запись единицы в регистр DI

```
_asm mov di, 1
```

не привела бы в нашем случае к желаемому результату, поскольку функция **_chain_intr()** все равно восстановила бы из стека значение регистра DI, каким оно было в точке прерывания, и затерла записанную нами единицу.

Функция **_chain_intr()** обычно используется в тех случаях, когда определенный пользователем новый обработчик прерывания начинает работу, а затем передает управление старому обработчику для ее завершения.

2. Вызов старого обработчика посредством оператора функции ()

Вызов старого обработчика прерывания осуществляется оператором вызова функции ():

```
(*oldint)();
```

или, как допускает теперь стандарт ANSI, просто

```
oldint();
```

Осуществляя такой вызов, следует учитывать тот факт, что если старый обработчик прерывания является обработчиком программного прерывания (например, INT 10H, INT 16H, INT 21H), то он наверняка изменит значения некоторых регистров, в которые он поместит результаты своей деятельности. Однако по окончании своей работы функция **newint()** восстановит из стека значения всех регистров, затерев тем самым плоды труда старого обработчика. Чтобы этого не произошло, необходимо скорректировать содержимое стека, то есть присвоить псевдорегистрам значения соответствующих регистров.

Фирма Microsoft в демонстрационной программе ALARM.C, входящей в On-line Help, предлагает следующее решение этой проблемы (пример 7).

Пример 7. Реализация функции обработки прерываний в демонстрационном примере фирмы Microsoft

```
/* Функция предотвращает повторный вызов прерывания 10H, если
* оно уже выполняется. Вероятность такого события мала, но оно
* может привести к непредсказуемым последствиям, если обработ-
* чик был прерван в момент доступа к аппаратным регистрам. */
void _interrupt _far newvideo( unsigned _es, unsigned _ds,
                               unsigned _dl, unsigned _sl,
                               unsigned _bp, unsigned _sp,
                               unsigned _bx, unsigned _dx,
                               unsigned _cx, unsigned _ax,
                               unsigned _ip, unsigned _cs,
                               unsigned _flags )

{
  static unsigned save_bp;

  /* Если мы не находимся в прерывании, то передадим
  * управление старому обработчику. */
  if( !finvideo )
      _chain_intr( oldvideo );
  else
  {
      /* Установим флаг занятости и восстановим значения
      * всех регистров, какими они были в точке прерывания.
      * Некоторые из них могли быть изменены предшествующим
      * кодом. Регистр BP должен быть сохранен в статической
      * переменной, для того чтобы он мог быть восстановлен
      * без модификации стека. */
      ++finvideo;
      _asm
      {
          mov     ax, _ax
          mov     bx, _bx
          mov     cx, _cx
          mov     dx, _dx
          mov     es, _es
          mov     dl, _dl
          mov     save_bp, bp
          mov     bp, _bp
      }

      /* Вызовем старый обработчик прерывания. */
      (*oldvideo)();

      /* Запишем в псевдорегистры значения соответствующих
      * регистров. Сбросим флаг занятости. */
      _asm
      {
          mov     bp, save_bp
          mov     _bp, bp
          mov     _dl, dl
          mov     _es, es
          mov     _dx, dx
          mov     _cx, cx
          mov     _bx, bx
          mov     _ax, ax
      }
      --finvideo;
  }
}
```

К сожалению, программа ALARM.C содержит ошибку (надеюсь, что этот факт не пробудит в вашей душе восторга «сибирского лесоруба»). Флаг finvideo инициализирован нулевым значением, поэтому условие !finvideo всегда будет истинным и управление будет передаваться старому обработчику прерывания. Таким образом, часть else никогда не сможет получить управление, а флаг finvideo навеки останется равным нулю. О проблеме нереентерабельности функций BIOS и DOS мы поговорим позже. Отметим пока, что в программе ALARM.C она не была решена. То, ради чего был задуман новый обработчик прерывания 10H — поддержание флага finvideo во взведенном состоянии на протяжении выполнения обслуживающих функций BIOS для работы с экраном — не было реализовано. Как следствие — некорректная ра-

бота программы в случае подачи сигнала в момент, когда прерывание INT 10H еще не завершило свою работу. В этом нетрудно убедиться. Запустите ALARM, указав в командной строке время, на минуту отличающееся от текущего, и дайте команду вывода на консоль какого-нибудь большого текстового файла. С большой вероятностью, после того как раздастся звуковой сигнал, Ваш компьютер зависнет. Если же этого не произойдет, не беда, попробуйте еще раз.

На первый взгляд может показаться, что исправить допущенную в программе ALARM.C ошибку очень легко. Достаточно лишь изменить условие **if (!finvideo)** на **if (finvideo)**, заново скомпилировать программу, запустить ее на выполнение и... убедиться, что Ваш компьютер ведет себя непонятным образом. В чем же дело? Неужели предложенный фирмой Microsoft способ взаимодействия со старым обработчиком прерывания нам не подходит? Увы, да. Причин здесь несколько. Остановимся лишь на одной из них.

Настройка регистра DS, выполняемая в начале каждого обработчика прерывания, дает возможность обращаться к глобальным переменным С программы непосредственно из обработчика. С другой стороны, потеря истинного значения DS, каким оно было в точке прерывания, приводит к тому, что старому обработчику будет передано совсем не то значение DS, которое он, очевидно, ожидает. Для аппаратного прерывания (INT 8H, INT 9H) такая потеря не существенна, а вот для программного прерывания это может привести к непредсказуемым последствиям.

Остается только сожалеть, что фирма Microsoft, реализовав очень полезную функцию **_chain_intr()** (фирма Borland не сделала и этого), не предложила еще одну функцию, назовем ее, скажем, **_call_intr()**, предназначенную для корректного обращения к старому обработчику прерывания. Что ж, поскольку название уже придумано, восполним пробел, допущенный фирмой Microsoft. Листинг 3 содержит текст функции **_call_intr()**, написанной на ассемблере и реализованной с учетом особенностей компиляторов обеих фирм. Там же приведен и слегка измененный текст функции **_chain_intr()**. Тексты программ хорошо документированы, поэтому для программистов, знакомых с ассемблером, не составит никакого труда разобраться в них. Впрочем, это и не обязательно. Мы же пользуемся библиотечной функцией **_chain_intr()** да и другими функциями, не вникая в подробности их реализации, но зная, как они работают.

Для ассемблирования файла CALLINTR.ASM можно использовать Microsoft Assembler, Turbo Assembler или же Quick C 2.51. Чтобы процесс ассемблирования завершился успешно, необходимо определить символические константы MSC или BC в зависимости от того, в какой среде будут использоваться функции **_chain_intr()** и **_call_intr()**. Сделать это можно следующим образом:

```
MASM -ML -DMSC CALLINTR.ASM
TASM -ML -DBC CALLINTR.ASM
```

Описание функции **_call_intr()** начнем с ее прототипа:

```
void _call_intr(void(_interrupt _far *)());
```

Функция **_call_intr()**, так же как и **_chain_intr()**, является программным трюком и гарантирует корректную работу только при условии, что ее вызов осуществляется из обработчика прерывания, написанного на С.

Получив в качестве аргумента адрес старого обработчика прерывания, функция сохраняет в стеке текущие значения всех регистров и флагов, затем, как и функция **_chain_intr()** загружает все регистры значениями, хранящимися в соответствующих псевдорегистрах, и, сбросив флажки TF и IF, подобно тому, как это делает микропроцессор, передает управление старому обработчику прерывания. По окончании его работы функция **_call_intr()** записывает значения всех регистров в соответствующие псевдорегистры, затем восстанавливает сохраненные в стеке значения регистров и флагов, какими они были в момент вызова **_call_intr()**, и возвращает управление вызвавшему ее новому обработчику прерывания.

Выяснив, как работает функция **_call_intr()**, перепишем приведенный выше обработчик прерывания 10H (пример 8). Заметим, что теперь функция **newvideo()** не нуждается в формальных параметрах.

Пример 8. Новый вариант обработчика прерывания вектора 10h

```
void _interrupt _far newvideo( void )
{
  ++finvideo;
  _call_intr( oldvideo );
  --finvideo;
}
```

Существуют, однако, несколько случаев, в которых функцию **_call_intr()** вызывать нельзя. Это не является ее недостатком, а связано с особенностью реализации некоторых обслуживающих функций DOS. Таких функций пять, и они должны вызываться только при помощи **_chain_intr()**:

00h — завершить программу;
31h — завершить программу и оставить ее резидентной;
4ch — завершить программу с кодом завершения (все три функции не возвращаются в точку их вызова);
26h — построить PSP (использует адрес вызвавшей ее функции);
4bh — передать управление другому процессу (может разрушить значения регистров при возврате).

Поэтому обработчик прерывания INT 21H должен выглядеть примерно следующим образом (пример 9).

Пример 9. Обработчик прерывания Int 21h

```
#define highbyte(a) (((a) > > 8) & 0x00ff)

void _interrupt _far newdos( ARGS )
{
  switch ( highbyte( _ax )) {
        case 0x26:
        case 0x4b:
        case 0x00:
        case 0x31:
        case 0x4c:
              _chain_intr( olddos );
  }

  _call_intr( olddos );
}
```

Вот и все, что нам необходимо знать о функции **_call_intr()**. Теперь, с учетом изложенного, исправим

ошибку в файле ALARM.C и после компиляции программы попробуем ею воспользоваться. Не нужно быть излишне требовательным, чтобы обратить внимание на такие ее недостатки, как

- отсутствие контроля повторной загрузки;
- отсутствие возможности выгрузки программы из памяти;
- отсутствие возможности корректировки установленного времени;
- непривычный синтаксис задания времени и отсутствие контроля допустимости введенного значения;
- слишком большой размер EXE файла для такой простой программы.

О том, как с ними бороться, мы поговорим в следующем номере нашего журнала. ■

Литература

1. Ю-Чжен Лю, Г.Гибсон. Микропроцессоры 8086/8088.– М.: Радио и связь, 1987.
2. Р.Джордейн. Справочник программиста персональных компьютеров типа IBM PC, XT и AT.– М.: Финансы и статистика, 1991.
3. Н.А.Войников. Системное программирование для ПРАВЕЦ-16.– София: Техника.
4. А.Рыскунов. Этот безумный, безумный, безумный мир резидентных программ// Компьютер Пресс.– 1992.– № 4.

Лаборатория программных средств "Shift" предлагает
семейство продуктов

Data Vision ™, *v 1.1*

Объектно - ориентированная библиотека для Turbo Pascal v6.0–7.0 и Borland Pascal, позволяющая реализовать объектный интерфейс приложений по стандартам CUA с **любыми** реляционными базами данных.

Файлы dBase, Paradox, Clarion и Ваши оригинальные форматы в одной таблице одновременно и по отдельности в DOS!

Соединение гибкости Turbo Vision и мощи реляционных баз данных.

Доступные цены, гарантии, печатная документация, электронный справочник, поддержка **UpGrade, Hot–Line** и возможная поставка исходных текстов.

Эксклюзивный дилер в г. Москва
Журнал "МОНИТОР" тел.: 238–99–33

г. Тамбов–23, а/я 115, ЛПС "SHIFT", тел.:(0752)–22–44–63

Миллионы наших зрителей – это миллионы Вашей прибыли!

Информационное агентство
РОСТ-Информ
разместит Вашу рекламу в передачах телекомпании РОСТ, работающей на 30-м канале ДМВ для 2 миллионов жителей Тульской области.

Р	☎
РОСТ-ИНФОРМ	33-15-55
300600, г. Тула, ул. Менделеевская, 1	33-15-56

Вместе — вглубь России!

Листинг 1

```
/* Исходный файл      : mtsrsize.c
* Компилятор         : Microsoft C 6.0, Quick C 2.51
* Модель данных      : Small */

#include <dos.h>
#include <stdlib.h>

enum { ERROR = -1, OK };

/* размер выделенной динамической памяти в параграфах */

#define HEAP_SIZE 64

/* дальние указатели заставляют компилятор выполнять разделение
на сегмент/смещение автоматически */

char _huge    *tsrstack;
char _huge    *tsrbottom;

int main (void)
{
    unsigned tsrsize;

    /* Инициализация стека и определение адреса конца программы */
    _asm mov WORD PTR tsrstack[0], sp
    _asm mov WORD PTR tsrstack[2], ss
    FP_SEG( tsrbottom) = _psp;
    FP_OFF(tsrbottom ) = 0;

    /* Размер программы определяется следующими параметрами :
    *     верхушка стека
    *     - нижняя граница программы (преобразованная в параграфы )
    *     + размер выделенной динамической памяти
    *     + один лишний параграф для учета остатка
    *     при целочисленном делении */
    tsrsize = ((tsrstack-tsrbottom) > > 4) + HEAP_SIZE + 1;

    /* Конец работы с сохранением в памяти
    резидентной части программы */

    _dos_keep( OK, tsrsize);

    /* Сюда мы попасть не должны! */
    return ERROR;

}
```

Листинг 2

```
/* Исходный файл      : btsrsize.c
* Компилятор         : Turbo C++ 1.0, Borland C 2.0
* Модель данных      : Small */

#include <dos.h>
#include <stdlib.h>

enum { ERROR = -1, OK };

/* размер выделенной динамической памяти и стека в параграфах */

unsigned _stklen    = 256;
unsigned _heaplen   = 1024;

/* дальние указатели заставляют компилятор выполнять разделение
на сегмент/смещение автоматически */

char _huge    *tsrstack;
char _huge    *tsrbottom;

int main (void)
{
    unsigned tsrsize;

    /* Инициализация стека и определение адреса конца программы */
    tsrstack    = MK_FP( _SS, _SP);
    tsrbottom = MK_FP( _psp, 0);

    /* Размер программы определяется следующими параметрами :
    *     верхушка стека
    *     - нижняя граница программы (преобразованная в параграфы )
    *     + один лишний параграф для учета остатка
          при целочисленном делении */
    tsrsize = ((tsrstack - tsrbottom) > > 4) + 1;

    /* Конец работы с сохранением в памяти
    резидентной части программы */

    keep( OK, tsrsize);

    /* Сюда мы попасть не должны! */
    return ERROR;

}
```

Листинг 3

```
; CALLINTR.ASM
; Автор: Дмитрий Валдин.
    .model small, c
; Последующие определения используются для
; обеспечения доступа к формальным параметрам
; пользовательской процедуры обработки прерываний
IFDEF MSC
    _flags    equ Word Ptr [bp+24]
    _ax       equ Word Ptr [bp+18]
    _cx       equ Word Ptr [bp+16]
    _dx       equ Word Ptr [bp+14]
    _bx       equ Word Ptr [bp+12]
    _bp       equ Word Ptr [bp+8]
    _si       equ Word Ptr [bp+6]
    _di       equ Word Ptr [bp+4]
    _ds       equ Word Ptr [bp+2]
    _es       equ Word Ptr [bp+0]
ELSEIFDEF BC
    _flags    equ Word Ptr [bp+22]
    _ax       equ Word Ptr [bp+16]
    _bx       equ Word Ptr [bp+14]
    _cx       equ Word Ptr [bp+12]
    _dx       equ Word Ptr [bp+10]
    _es       equ Word Ptr [bp+8]
    _ds       equ Word Ptr [bp+6]
    _si       equ Word Ptr [bp+4]
    _di       equ Word Ptr [bp+2]
    _bp       equ Word Ptr [bp+0]
ENDIF

    .CODE
IFDEF BC
    PUBLIC _call_intr, _chain_intr
ENDIF
_call_intr    PROC NEAR
              push    bp        ; Шаг 1: формирование стекового кадра.
              mov     bp, sp    ; Регистр BP будет восстановлен на шаге 3.
```

Листинг 3 (продолжение)

```
                mov     cx, Word Ptr [bp+6]     ; Шаг 2: сохранить вектор
                                                ; в паре CX:DX.
                mov     dx, Word Ptr [bp+4]     ; Эти значения будут
                                                ; использованы на шаге 7.

                pop     bp          ; Шаг 3: восстанавливаем BP.

                push    si          ; Шаг 4: сохраняем регистры, важные
                push    di          ; для среды языка C.
                push    ds
                push    es

                push    bp          ; Шаг 5: сохраняем также BP и флаги.
                pushf

                mov     ax, _flags ; Подготовка к вызову обработчика
                push    ax
                push    cs          ; Шаг 6: поместить в стек все объекты,
                mov     bx, offset ret_addr  ; которые необходимы
                                    ; для работы обработчика,
                push    bx          ; а также регистр флагов и
                                    ; адрес возврата (CS:ret_addr).

                and     ah, 0fch    ; Шаг 7: сбросить флаги TF и IF
                push    ax          ; и поместить в стек регистр флагов
                push    cx          ; и адрес первоначального обработчика.
                push    dx

                mov     ax, _ax     ; Шаг 8: загрузка регистров.
                mov     cx, _cx     ; BP загружается в последнюю очередь.
                mov     dx, _dx
                mov     bx, _bx
                mov     si, _si
                mov     di, _di
                mov     ds, _ds
                mov     es, _es
                mov     bp, _bp

                iret                ; Шаг 9: загрузить флаги и передать
                                    ; управление обработчику. Команда
                                    ; IRET использует результаты усилий,
                                    ; приложенных на шаге 7.

ret_addr:
                                    ; В эту точку мы попадаем по комаиде
                                    ; IRET, и в стекс остается то,
                                    ; что было загружено на шаге 6.
                                    ; (Конечно, если обработчик прерываний
                                    ; не внес в стек никаких изменений!)

                push    bp          ; Шаг 10: временное сохранение
                pushf               ; возвращаемых значений BP и флагов.

                                    ; Шаг 11: загрузка BP для обновления
                                    ; значений формальных параметров
                                    ; процедуры обработки прерываний
                mov     bp, sp
                mov     bp, Word Ptr [bp+6]

                mov     _ax, ax     ; Шаг 12: собственное обновление
                mov     _cx, cx     ; формальных параметров.
                mov     _dx, dx
                mov     _bx, bx
                mov     _si, si
                mov     _di, di
                mov     _ds, ds
                mov     _es, es

                pop     _flags      ; Шаг 13: восстановление значений,
                pop     _bp         ; помещенных в стек на шаге 10.

                popf                ; Шаг 14: восстановление флагов и BP,
                pop     bp          ; сохраненных на шаге 5.

                pop     es          ; Шаг 15: восстановление регистров,
                pop     ds          ; используемых средой C
                pop     di          ; (смотри шаг 4).
                pop     si

                ret                 ; Шаг 16: возврат управления вызывающей
                                    ; процедуре.
_call_intr ENDP

_chain_intr     PROC NEAR

; Эта процедура является, по существу, программным трюком.
; Но она работоспособна, поскольку мы знаем, как Borland и Microsoft
; осуществляют сохранение регистров при вызове функции _interrupt.

                push    bp          ; Шаг 1: установка стекового кадра.
                mov     bp, sp      ; Значение BP будет восстановлено
                                    ; на шаге 3.

IFDEF MSC                           ; Шаг 2: загрузка вектора прерываний в
                mov     cx, Word Ptr [bp+6]     ; cx:dx или bx:cx пару.
                                                ; Выбор определяется
                mov     dx, Word Ptr [bp+4]     ; используемым
                                                ; компилятором.
                                                ;Эти данные
ELSEIFDEF BC                                    ; будут использованы
                                                ; на шаге 8.
                mov     bx, Word Ptr [bp+6]
                mov     cx, Word Ptr [bp+4]
ENDIF

                pop     bp          ; Шаг 3: восстановление стекового кадра.
                mov     sp, bp      ; Так мы можем получить доступ к сохра-
                                    ; ненным значениям регистров.
                                    ; Шаг 4: загрузка в AX регистра флагов,
                                    ; сброс IF и TF.
                mov     ax, _flags
                and     ah, 0fch
                                    ; Шаг 5: восстановление сохраненных
                xchg    ax, _ax     ; регистров.
                xchg    cx, _cx
IFDEF MSC
                xchg    dx, _dx
                pop     es
                pop     ds
                pop     di
                pop     si
                pop     bp
                pop     bx
                pop     bx
ELSEIFDEF BC
                xchg    bx, _bx
                pop     bp
                pop     di
                pop     si
                pop     ds
                pop     es
                pop     dx
ENDIF                               ; Шаг 6: передача управления
                                    ; первоначальному обработчику.
                iret
_chain_intr     ENDP
                END
```

ПРИВАТИЗИРУЕМЫЕ ПРЕДПРИЯТИЯ И АКЦИОНЕРНЫЕ ОБЩЕСТВА!

Фирма ИНФОРКОМ утверждает, что выпуск ценных бумаг и многотомных списков акционеров является дорогим и неэффективным делом. Гораздо проще приобрести

Программный комплекс ДЕПОЗИТАРИЙ

Приобретая наш программный комплекс, Вы получаете в руки мощное и простое в эксплуатации средство для проведения безналичной эмиссии и учета движения разнообразных ценных бумаг.

ПК ДЕПОЗИТАРИЙ позволит Вам решить большинство Ваших проблем.

Формирование базы данных инвесторов и статистических справок.
Вы получаете широкие возможности для сортировки, поиска, статистической оценки информации и печати списков инвесторов в требуемой Вам форме.

Ведение истории покупок для каждого инвестора.
Вы получаете возможность в любой момент узнать, когда, в каком количестве и по какой цене каждый инвестор покупал ценные бумаги.

Автоматический рассчет дивидендов.
Вы получаете возможность расчитывать дивиденды в любой момент.

Одновременное ведение реестров различных эмитентов.
Вы получаете возможность на коммерческой основе обслуживать одновременно несколько акционерных обществ, предоставляя им разнообразные дополнительные услуги.

Организация брокерской сети.
Вы получаете возможность вести центральный реестр и собирать в него информацию с рабочих мест брокеров через систему электронной почты или на физических носителях.

В настоящее время ПК ДЕПОЗИТАРИЙ успешно эксплуатируется в АО «ЭФФИ», обслуживающем несколько крупных эмитентов.

Условия поставки
В минимальный комплект поставки входят центральный реестр и одно рабочее место брокера. Количество дополнительно поставляемых рабочих мест брокера не ограничено.

Звоните: (095) 452-31-03 (вторник, среда, четверг).
Пишите: 121019 Москва, Г-19, а/я 16, ИНФОРКОМ

Показать картинку

А. Тихонов

Часть 2

В предыдущей статье [1] мы обсуждали, как показать картинку в формате PCX. Теперь рассмотрим формат Graphic Interchange Format (GIF), предложенный фирмой Compuserve для использования в информационных сетях. Это, естественно, наложило на него определенный отпечаток: во-первых, по сравнению с PCX GIF гораздо меньше зависит от используемых технических средств, во-вторых, за счет использования LZW алгоритма [2] и побитовой упаковки значительно повышена степень сжатия графической информации, что существенно при передаче по низкоскоростным телефонным каналам, в-третьих, обеспечена возможность хранения нескольких картинок в одном файле и, наконец, при показе картинки введен так называемый режим чередования (interlace), дающий возможность судить о картинке в целом задолго до того, как весь файл передан по телефонной линии, и прервать передачу, если картинка не нужна.

Статья содержит программу визуализации картинок в формате GIF на Turbo Pascal, при этом используются только штатные средства системы программирования. Это обусловливает возможность показа картинок только в стандартных для Borland Graphic Interface режимах. Средства для упаковки картинок в формат GIF можно легко сконструировать, прочитав статью [2]. В книгах [3], [4] приведены программы на языках C и ассемблер, соответственно, ускоряющие визуализацию картинок с помощью непосредственного обращения к аппаратным средствам адаптеров EGA/VGA.

В настоящее время сосуществуют два стандарта Graphic Interchange Format — GIF87a и GIF89a. Последний отличается от GIF87a дополнительными возможностями по описанию технических средств. Стандарты совместимы сверху вниз, в данной статье обеспечена совместимость с GIF87a.

Структура GIF файла приведена на рис. 1. GIF файл включает в себя сигнатуру, используемую программами визуализации для опознавания формата GIF, описание требований к графическому оборудованию (логический экран), на котором будут демонстрироваться картинки (размеры экрана, число используемых цветов, цветовая палитра); далее следуют блоки с графическими образами, а также так называемые блоки расширения, служащие для передачи дополнительных данных программам визуализации. Тип блока идентифицируется по первому байту. Блоки, содержащие картинки, начинаются с байта $2C, блоки, содержащие дополнительные сведения, нашим визуализатором пропускаются. Признаком конца GIF файла служит байт-терминатор с шестнадцатиричным значением $3B.

В листинге 1 приводится модуль Gif и функция GifViewer, осуществляющая показ картинок. Функции передается единственный параметр — имя GIF файла, возвращает она код завершения, успешному показу картинки соответствует код 0.

Последовательность действий по показу картинок такова: открытие файла (функция

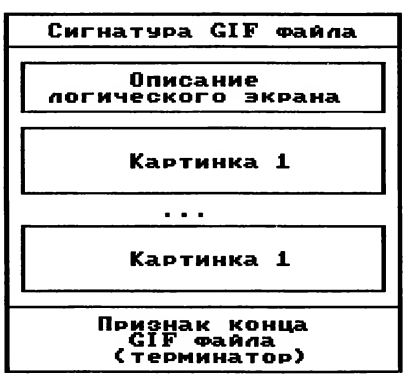


Рис. 1. Структура GIF файла

OpenGifFile, листинг 2); проверка сигнатуры (OkGifSignature); загрузка и проверка корректности описания логического экрана (GetScreenDescription). Далее осуществляется переход в графический режим (GifInitGraph). В цикле до появления признака конца GIF файла последовательно читаются и обрабатываются блоки данных. Чтение и обработка типа блока осуществляются функцией GetRecordType. Блоки расширения нашим визуализатором пропускаются, что допускается стандартом GIF87a.

В качестве сигнатуры используется строка с именем стандарта, то есть GIF87a или GIF89a, что и проверяется функцией OkGifSignature (листинг 2).

Описание логического экрана содержится в структуре ScrDsc, поля которой определяют его горизонтальный и вертикальный размеры и цвет фона. Размер структуры 7 байтов, заканчивается она нулевым байтом. В ScrDsc присутствует байтовое поле Flag, биты которого означают следующее. Старший бит указывает, используется ли глобальная таблица цветов, речь о которой пойдет далее. Если глобальная таблица цветов отсутствует, то в нашей программе используется по умолчанию таблица EgaColorMap (листинг 3). В младших трех битах поля Flag хранится информация о числе используемых цветов. Это битовое поле может принимать значения от 0 до 7. Нулю соответствуют два цвета, единице — четыре, трем — шестнадцать, семи — 256 цветов. Максимальное цветовое разрешение в Gif составляет 256 цветов.

Programmer

В стандарте GIF степень сжатия графической информации существенно повышена за счет использования LZW алгоритма и побитовой упаковки.

Поле BackGround в структуре ScrDsc представляет собой индекс цвета фона. Поясним, что это означает. Информация о цветовой палитре в Gif хранится в цветовых таблицах. В картинках каждый элемент изображения ссылается на вход таблицы. Вход содержит три байта: относительную яркость красного, зеленого и синего цвета в диапазоне от 0 до 255. Так, ярко-белый цвет соответствует входу цветовой таблицы (255,255,255), черный (0,0,0), а желтый (180,180,0). Цвет элемента изображения — пиксела в Gif кодируется номером входа в цветовую таблицу.

Здесь надо сказать несколько слов о воспроизведении картинок с большим цветовым разрешением (например, 256 цветов) на дисплеях с меньшим цветовым разрешением (16 цветов). В Gif87a не оговорен порядок расположения цветов и алгоритм трансляции их в реальные цвета. Однако при формировании GIF файла таблица может быть упорядочена так, чтобы первыми в ней стояли наиболее употребительные цвета, то есть те, которые наиболее часто используются в картинке. При этом цветопередача, естественно, улучшится.

Процедура GetScreenDescription осуществляет чтение описания логического экрана, разбирает его и формирует внутреннее описание в структуре Description. Глобальная цветовая таблица, если она присутствует, считывается из файла и также переписывается в Description.

При инициализации графики (функция GifInitGraph в листинге 2) дисплей переводится в графический режим с максимальным разрешением и проверяются условия воспроизведения картинки на экране.

Несколько слов необходимо сказать о структуре блоков расширения (функция SkipBlock). Первый байт содержит функциональный код, воспринимаемый обработчиком блока расширения. Далее следуют блоки данных, предваряемые байтом-счетчиком, содержащим длину очередного блока. Таким образом, блок расширения всегда заканчивается нулевым байтом.

Перейдем теперь к рассмотрению чтения и показа очередной картинки на экране дисплея. Каждая картинка в Gif предваряется описанием (структура ImageDsc в листинге 3). Описание картинки содержит координаты положения ее верхнего левого угла на логическом экране, ширину и высоту в пикселах. Старший (седьмой) бит байтового поля Flag указывает, используется ли локальная таблица цветов. Если он установлен в единицу, то непосредственно за структурой ImageDsc следует таблица цветов, относящаяся только к данной картинке. В противном случае используется глобальная таблица цветов. Следующий шестой бит является флажком, указывающим режим отображения картинки. Если он установлен в единицу, то используется режим чередования строк. В этом случае вывод картинки на экран осуществляется не строка за строкой, а в четыре прохода. На первом проходе отображается каждая восьмая строка, начиная с нулевой, на втором — каждая восьмая, начиная с пятой, на третьем — каждая четвертая, начиная со второй, и, наконец, на последнем — каждая вторая строка, начиная с первой. Пересчет номеров строк осуществляется функцией Interlace.

Младшие три бита в описании картинки точно так же, как и в описании логического экрана, определяют, сколько цветов используется при ее показе. Функция GetImageDescription считывает описание картинки, переписывает ее положение и размеры в структуру Description, формирует цветовую таблицу.

Чтение, преобразование сжатых растровых данных их GIF файла осуществляет функция GetRasterDateAndDisplay (листинг 3).

Растровые данные со сжатым изображением картинки предваряются байтом, содержащим минимальное число битов, используемых для кодирования пиксела (InitBitsPerPixel),которое обычно совпадает с числом битов, используемых для кодирования цвета в изображении.

Далее производится установка палитры EGA/VGA в соответствии с цветовой таблицей (процедура SetPaletteFromGif, листинг 4), инициализация данных для декодирования картинки (процедура InitDecoding). EGA/VGA позволяет устанавливать только четыре уровня интенсивности для каждого из основных цветов, поэтому SetPaletteFromGif аппроксимирует цветовую таблицу GIF и устанавливает палитру EGA/VGA.

В циклах по ширине и высоте картинки осуществляется получение очередного пиксела (функция GetPixelFromGif, листинг 4) и вывод его на экран (функция PutPixelToScreen). Вычисление номера очередной строки, в зависимости от того, установлен или нет режим чередования, осуществляется функцией Interlace.

Процедура InitCodeTable (листинг 5) инициализирует данные для использовании LZW алгоритма.

Теперь можно непосредственно обратиться к преобразованию графической информации. В стандарте GIF графическая информация организована в блоки. Ей предшествует, как уже говорилось, байт с начальным числом битов, используемых для кодирования элемента изображения — пиксела. Этот байт считывается в функции GetRasterDataAndDisplay и обычно совпадает с информацией в описании картинки. Исключением являются монохромные картинки — в них кодирование начинается не с одного, а с двух битов на пиксел.

Далее следуют блоки данных, каждому блоку предшествует байт-счетчик.

В стандарте GIF элементу изображения соответствует номер цвета в цветовой таблице. Последовательность элементов изображения-пикселов сжимается с помощью LZW алгоритма [2]. Полученная таким образом последовательность LZW кодов переменной длины дополнительно упаковывается. Например, для шестнадцатицветовой картинки кодирование можно начинать с 4 битов на пиксел, однако нужно помнить, что в GIF определены еще два дополнительных кода (код сброса цветовой таблицы CC и код окончания графической информации EOI), так что придется начинать кодирование с пяти битов на пиксел. По мере увеличения LZW таблицы, а она может содержать до 4096 кодов, размер кода увеличивается до 12 битов. Этот факт и обусловил то, что разработчики стандарта ввели дополнительную побитовую упаковку кодов.

При показе картинки действие развивается в обратном порядке. Очередной пиксел для вывода на экран извлекается функцией GetPixelFromGif (листинг 5). Эта функция выбирает очередной пиксел из буфера PixelBuffer, если он не пуст, в противном случае получает очередной LZW код от функции GetCode и преобразует этот код в цепочку пикселов, помещая последнюю в PixelBuffer. Здесь же ведется обработка специальных кодов CC и EOI. Отметим также, что пикселы в буфере хранятся в обратном порядке.

В этой же функции осуществляется пополнение LZW таблицы с помощью процедуры NewTableEntry. Там же отслеживается текущий размер кода (CurrentBitsPerPixel).

Декодирование LZW кодов совмещено с пополнением LZW таблицы. Каждый вход таблицы содержит последний пиксел цепочки и ссылку на предыдущий вход. Корень таблицы, содержащий коды цветов, а также CC и EOI, формируется процедурой InitDecoding. Для входов, содержащихся в корне таблицы, вместо ссылки на предыдущий вход кодируется -1.

Сам алгоритм декодирования представлен на рис. 2. Вся прелесть LZW алгоритма состоит в том, что из входной последовательности кодов извлекаются либо коды, уже имеющиеся в таблице, либо на единицу превышающие текущий размер в таблице. В первом случае последовательность пикселов легко декодируется извлечением очередного пиксела из таблицы и последующим переходом ко входу, указанному в ссылке. Декодирование кончается в том случае, когда очередная ссылка равна -1. Во втором случае в конец ранее декодированной цепочки пикселов дописывается ее первый пиксел и сформированная таким образом последовательность заносится в таблицу LZW кодов.

1. Инициализация буфера пикселов.
2. Получение текущего LZW-кода с помощью функции GetCode.
3. Декодирование текущего LZW кода и запись соответствующей ему цепочки в буфер пикселов.
4. Сохранение текущего LZW кода (CurrentCode) в переменной OldCode.
5. Если CurrentCode = EOI, то закончить декодирование.
6. CurrentCode := GetCode;
7. Если CurrentCode уже находится в таблице LZW кодов, то декодируется цепочка символов для CurrentCode. В таблицу LZW кодов добавляется цепочка, состоящая из первого символа декодированной цепочки и цепочки, соответствующей OldCode.
8. OldCode := CurrentCode.
9. Иначе
10. Декодируется последовательность для OldCode. Цепочка, соответствующая CurrentCode, формируется дописыванием первого пиксела в конец цепочки. Сформированная последовательность записывается в таблицу LZW кодов.
11. OldCode:=CurrentCode.
12. Переход на 11.

Рис. 2. Псевдокод алгоритма декодирования цепочки пикселов

Функция GetCode занимается распаковкой LZW кодов из последовательности байтов. В Turbo Pascal отсутствуют средства для работы с цепочками битов, поэтому приходится выделять коды с помощью сдвигов и битовых масок, хранимых в массиве BitMask. Очередной байт для распаковки GetCode получает от функции GetByte. В свою очередь функция GetByte осуществляет чтение байтовых блоков и их буферизацию.

В листинге 6 представлена программа, получающая имя GIF файла из командной строки и отображающая его на экране дисплея. Программа обрабатывает ряд ошибок, их перечень представлен в начале листинга 2.

В заключение отметим, что скорость работы предлагаемой программы можно значительно повысить (проверьте это с помощью Turbo Profiler), переписав на ассемблере всего две функции: распаковки LZW кодов и выдачи на экран пикселов. Во втором случае целесообразно обойтись без сравнительно медленной библиотеки BGI, непосредственно работая с видеопамятью. ■

Литература

1. Тихонов А. Показать картинку// Монитор.– 1992.– № 3.
2. Арапов Д. Пишем упаковщик// Монитор.– 1993.– № 1.
3. В.Ю.Романов. Популярные форматы файлов для хранения графических изображений.– М.: Унитех, 1992.
4. Rimmer S. Bit-Mapped Graphics.– USA: Windcrest, 1990.

Листинг 1. Модуль GIF

```
{A-,R-,D-,V-,B-}
unit gif; {listing1.pas}
{*************************************************}
{ unit GIF - визуализатор картинок в формате GIF87a }
{ (C) 1993, А. Тихонов                            }
{*************************************************}

Interface

uses Crt,Graph;

function GifViewer( GifFileName:String):Byte;
{ Основная процедура для показа GIF файла }

Implementation
var
     GrDriver,GrMode:Integer;
     SavedPalette: PaletteType;
          { Сохраненная палитра для EGA и VGA 16 цветов}

{$I listing2.pas}
{$I listing3.pas}
{$I listing4.pas}
{$I listing5.pas}

Function GifViewer( GifFileName:String):Byte;

var
     RecordType    :Byte;          { Тип блока в Gif файле}
     Palette       :PaletteType;   { Палитра Ega/Vga до показа картинки }

var
     Pos  : LongInt;
     c    : Char;
     b    : Byte;
begin
     GifViewer:= Ok;

     if (not OpenGifFile(GifFileName)) then begin
                    { Не смог открыть файл }
          GifViewer := GifError(Not_Opened);
          Exit;
     end;

     if (not OkGifSignature) then begin
          GifViewer := GifError(InCorrectGifHeader);
                    { Некорректный заголовок GIF файла }
          Exit;
     end;

     if GifInitGraph = GraphInitError then begin
          GifViewer :=
               GifError(GraphInitError);
                    { Не смог перейти в графический режим }
          Exit;
     end;

     b := GetScreenDescription;
     if ( b < > 0 ) then begin
          GifViewer := GifError(b);
                    { Некорректное описание логического экрана }
          Exit;
     end;

     {
     Теперь читаем и показывем картинки, содержащиеся в GIF файле
     }

     RecordType := ImageDescription; { Описание графического образа }

     repeat
          RecordType := GetRecordType;
          case RecordType of
          ImageDescription: begin        { Обработка и показ картинки }
               if (GetImageDescription < > 0) then begin
                    GifViewer:= GifError(InCorrectImageDescription);
                         { Некорректное описание графического образа }
                    Exit;
               end;
               GetPalette(Palette);       { Сохранение палитры }
               b := GetRasterDataAndDisplay;
               { Считывание данных из GIF файла и преобразование их
               в графический образ }
               if b < > 0 then begin
                    GifViewer:= GifError(b);
                    Exit;
               end;
               c:= ReadKey;               { Ожидание реакции пользователя}
               SetallPalette(Palette);    { Восстановление палитры }
          end;{ImageDescription}

          ExtensionBlock: begin     { Блоки расширения пропускаются }
               b := SkipBlock;
               if b < > 0 then begin
                    GifViewer:= GifError(b);
                    Exit;
               end;
          end;

          IncorrectRecordType: begin
               GifViewer:= GifError(IncorrectBlockType);
               Exit;
          end;

          Terminator: TerminateGif;     { Завершение работы с GIF файлом }

          0:;                           {Обработка конца блока}

          else begin
               GifViewer:= GifError(RecordType);
               Exit;
          end; {else}
          end;{case}

     until (RecordType = Terminator ) ;

     {$I-}
     Close(F);
     {$I+}

     if IoResult < > 0 then
          GifViewer:= GifError(CloseGifFileError);
                                   {Ошибка при закрытии файла }
     end;
end.
```

Листинг 2. Процедуры и функции, обеспечивающие чтение и показ картинок

```
{Файл listing2.pas}

const
     Ok                        = 0;   { Все в порядке }
     Not_Opened                = 1;   { Ошибки при открытии файла }
     IncorrectGifHeader        = 2;   { Некорректный заголовок
                                        GIF файла }
     InCorrectScreenDescription = 3;  { Некорректное описание
                                        логического экрана }
     IncorrectBlockType        = 4;   { Некорректный тип блока }
     CloseGifFileError         = 5;   { Ошибка при закрытии файла }
     GraphInitError            = 6;   { Ошибка при инициализации
                                        графики }
     InCorrectImageDescription = 7;   {Ошибка при чтении
                                        описания графического образа }
     InitBitsPerPixelError     = 8;   { Ошибка при чтении начального
                                        числа битов для кодирования
                                        одного пиксела }
     ByteBlockReadError        = 9;   { Ошибка при чтении
                                        блока кодов }
     TooLargeScreen            = 10,  { Слишком большой экран }
     TooManyColors             = 11;  { Слишком много цветов }
     ExtendedBlockError        = 12;  { Ошибка при чтении
                                        расширенного блока }
     ImageDataDecriptionError  = 13;  { Ошибка при чтении
                                        и обработке картинки }

const
     ImageDescription          = $2C; { Описание графического
                                        образа }
     Terminator                = $3B; { Окончания GIF файла }
     ExtensionBlock            = $21; { Блок расширения }
     IncorrectRecordType       = $FF; { Некорректный тип блока }

type
     RGB = (Red, Green, Blue);        { Основные цвета }

const
     EgaColorMap: array [0..15,Red..Blue] of Byte =
                                   {Цветовая палитра по умолчанию}
                    (
                         ( 0, 0, 0),      { 0. Black }
                         ( 0, 0,170),     { 1. Blue }
                         ( 0,170, 0),     { 2. Green }
                         ( 0,170,170),    { 3. Cyan }
                         (170, 0, 0),     { 4. Red }
                         (170, 0,170),    { 5. Magent }
                         (170,170, 0),    { 6. Brown }
                         (170,170,170),   { 7. LightGray }
                         ( 85, 85, 85),   { 8. DarkGray }
                         ( 85, 85,255),   { 9. LightBlue }
                         ( 85,255, 85),   {10. LightGreen }
                         ( 85,255,255),   {11. LightCyan }
                         (255, 85, 85),   {12. LightCyan }
                         (255, 85,255),   {13. LightMagenta }
                         (255,255, 85),   {14. Yellow }
                         (255,255,255)    {15. White }
                    );
```

Листинг 2 (продолжение)

```
type
    ColorMap = array[0..255,Red..Blue] of Byte;
                                    { Таблица описаний цветов }
    GifDescription = record
        ScreenWidth         :Integer;
                { Горизонтальный размер логического экрана }
        ScreenHeight        :Integer;
                { Вертикальный размер логического экрана }
        ImageLeft,
        ImageTop            :Integer;
                { Положение очередного образа на логическом экране }
        ImageWidth          :Integer;
                { Горизонтальный размер очередного образа }
        ImageHeight         :Integer;
                { Высота очередного образа }
        HasGlobalColorMap   :Boolean;
                { Наличие цветовой таблицы для описания экрана }
        BackGround          :Byte;
                { Цвет фона }
        ColorResolution     :Byte;
                {Число битов на пиксел }
        Colors              :Word;
                { Число цветов для логического экрана }
        ImageColors         :Word;
                { Число цветов для очередного образа }
        Interlace           :Boolean;
                { Нужно ли использовать "переплетение"
                для показа очередного образа }
        BitsPerPixel        :Byte;
                {Текущее число пикселов, используемое
                для LZW кодирования }
        GlobalColorMap      :ColorMap;
                { Описание цветов для логического экрана }
        LocalColorMap       :ColorMap;
                { Описание цветов для очередного образа }

    end; {GifDescription}

type
    ByteBuffer = Array[1..255] of Byte;

var
    IoError     :Integer;       { Признак ошибки ввода-вывода }
    F           :File;          { Файл с каратинками }
    Description :GifDescription;    { Внутренне описание
                                    логического экрана }

Function ReadByte(var IoError:Integer):Byte; Forward;
                { Чтение байта из GIF файла }

Procedure ReadByteBlock(Counter:Byte;var Buf:ByteBuffer;
                        var IoError:Integer); Forward;
                { Чтение блока данных из GIF файла}

Function GifError(Error:Integer):Byte;      { Обработка ошибок }
begin
    GifError := Error;
    If Error < > 0 then begin
        case Error of
            1 : writeln('Ошибка при открытии GIF файла');
            2 : writeln('Некорректный заголовок GIF файла');
            3 : writeln('Некорректное описание логического экрана');
            4 : writeln('Некорректный тип блока');
            5 : writeln('Ошибка при закрытии GIF файла');
            6 : writeln('Ошибка при инициализации
                    графического режима');
            7 : writeln('Некорректное описание картинки');
            8 : writeln('Некорректное значение числа символов,
                    используемых для кодирования пиксела');
            9 : writeln('Ошибка при чтении блока');
            10 : writeln('Слишком большое изображение');
            11 : writeln('Слишком много цветов');
            12 : writeln('Ошибка при чтении расширенного блока');
            13 : writeln('Ошибка при чтении и обработке картиники');
            else writeln('Ошибка #',Error);
        end;
    end;
end;

Function OpenGifFile(GifFileName:String):Boolean;
                                        { Открытие GIF файла }
begin
    {$I-}
    Assign(F,GifFileName);
    Reset(F,1);
    OpenGifFile := (IoResult = 0);
    {$I+}
end;

Function OkGifSignature:Boolean;        { Чтение и проверка сигнатуры }
var
    Signature:array[1..6] of char;
    Count     :word;
begin
    OkGifSignature := FALSE;
    {$I-}
    BlockRead(F,Signature,sizeof(Signature),Count);
    if   ((IOResult = 0) and
        (Count = sizeof(Signature)) and
        (not eof(F))) then
            OkGifSignature :=
                (Signature[1] = 'G') and
                (Signature[2] = 'I') and
                (Signature[3] = 'F') and
                (Signature[4] = '8') and
                (Signature[6] = 'a');
    {$I+}
end;

Function SkipBlock: Byte;               { Пропуск расширенного блока }
var
    B    :Byte;
    Buf  :ByteBuffer;
begin
    SkipBlock := 0;
    B := ReadByte(IoError); { Функциональный код расширенного блока }
    if IoError < > 0 then begin
        SkipBlock := B;
        Exit;
    end;

    B := ReadByte(IoError);                 { Размер блока данных }
    if IoError < > 0 then begin
        SkipBlock := ExtendedBlockError;
        Exit;
    end;

    while (B > 0) and (IoError = 0) do begin
        ReadByteBlock(B,Buf,IoError);
        if IoError < > 0 then SkipBlock := ExtendedBlockError
        else begin
            B := ReadByte(IoError);     { Размер блока данных }
            if IoError < > 0 then SkipBlock := ExtendedBlockError;
        end;
    end;
end;

Function GetScreenDescription :Byte;
                { Чтение и проверка описания логического экрана }

type
    ScrDsc = record
        Width       :Integer;  { Ширина логического экрана }
        Height      :Integer;  { Высота логического экрана }
        Flag        :Byte;     { Флажок }
        Background  :Byte;     { Цвет фона }
        Terminator  :Byte;     { Признак конца блока }
    end;

var
    ScreenDescription  :ScrDsc;
    Count, Size, D, i  :Word;
    J                  :RGB;
    C                  :Word;
begin
    GetScreenDescription := 0;

    {$I-}
    Size:=SizeOf(ScreenDescription);
    BlockRead(F,ScreenDescription,Size,Count);
    {$I+}

    if (IoResult < > 0) and (Size < > Count) then begin
        GetScreenDescription := InCorrectScreenDescription;
        Exit;
    end
    else
    if (Size = Count) then begin    { Формирование глобального описания}
        Description.ScreenWidth := ScreenDescription.Width;
        Description.ScreenHeight := ScreenDescription.Height;
        Description.BackGround := ScreenDescription.BackGround;
        Description.HasGlobalColorMap := (ScreenDescription.Flag and
                                        $80) < > 0;
                                    { Проверка установки старшего бита }
        Description.ColorResolution:= ((ScreenDescription.Flag and $70)
                                        shr 4)+1;
                                    { Выделяем 5..7 биты, сдвигаем их вправо
                                    и прибавляем единицу }
        D:= ScreenDescription.Flag and $07;
                                    {Выделяем три младших бита }
        D:= 1 shl (D+1);            { 2 в степени (D+1) }
        Description.Colors := D;
        Description.ImageLeft := 0;
        Description.ImageTop := 0;
        Description.ImageWidth := ScreenDescription.Width;
        Description.ImageHeight := ScreenDescription.Height;
        if (Description.ScreenWidth > (GetMaxX+1)) or
        (Description.ScreenHeight > (GetMaxY+1)) then begin
            GetScreenDescription := TooLargeScreen;
            Exit;
```

```
            end
            else
            if (Description.Colors > (GetMaxColor + 1)) then begin
                GetScreenDescription := TooManyColors;
                Exit;
            end
            else
            if Description.HasGlobalColorMap then begin
                for i:= 0 to Description.Colors-1 do
                for j := Red to Blue do
                    Description.GLobalColorMap[i,j]:= 0;
                Size := Description.Colors * 3 * sizeof(RGB);

                {$I-}
                BlockRead(F,Description.GlobalColorMap,Size,Count);
                {$I+}

                If (IoResult < > 0) or (Size < > Count) then begin
                    GetScreenDescription := InCorrectScreenDescription;
                end;
                for i:= 0 to Description.Colors-1 do
                for j := Red to Blue do
                    Description.LocalColorMap[i,j] :=
                        Description.GlobalColorMap[i,j];
            end {if}
            else begin { Доформировываем глобальную цветовую таблицу }
                C:= Description.Colors;
                if C > 16 then C:= 16;
                            { Больше 16 цветов не предусмотрено}
                for i:= 0 to C-1 do
                for j := Red to Blue do
                    Description.GlobalColorMap[i,j] := EgaColorMap[i,j];
                for i:= 0 to Description.Colors-1 do
                for j := Red to Blue do
                    Description.LocalColorMap[i,j] :=
                        Description.GlobalColorMap[i,j];
            end; {else}
        end;
    end;
```

Листинг 3. Считывание, обработка и демонстрация картинки

```
{Файл listing3.pas}

var
    InitBitsPerPixel    :Byte;
    { Начальное число битов, используемое для кодирования пиксела }
    CC                  :Byte;    { Признак очистки таблицы Lzw кодов }
    EOI                 :Byte;    { Признак конца картинки }

Procedure SetPaletteFromGif; Forward;
                { Установка палитры EGA/VGA }
Procedure InitDecoding; Forward;
                { Инициализация данных }
Function InterLace(y:Word):Word; Forward;
                { Вычисление номера строки в режиме чередования }
Function GetPixelFromGif(var IoError:Integer):Integer; Forward;
                { Извлечение очередного пиксела из GIF файла}
Procedure InitCodeTable; Forward;
                { Инициализация таблицы LZW кодов }
Procedure PutPixelToScreen(x,y,W:Word); Forward;
                { Вывод очередного пиксела }

Function GetScreenImageDescription:Byte;
                { Чтение и проверка описания графического образа }

type
    ImageDsc = record
        ImageLeft,
        ImageTop        :Integer;  { Левый верхний угол }
        ImageWidth,
        ImageHeight     :Integer;  { Ширина, высота образа}
        Flag            :Byte;     { Флажок }
    end;

var
    ImageDescription    :ImageDsc;
    Count, Size, D, i   :Word;
    j                   :RGB;
    HasLocalColorMap    :Boolean;
begin
    GetScreenImageDescription := 0;
    {$I-}
    Size := SizeOf(ImageDescription);
    BlockRead(F,ImageDescription,Size,Count);
    If (IoResult < > 0) or (Size < > Count) then begin
        GetScreenImageDescription := IncorrectImageDescription;
        Exit;
    end;
    {$I+}

    if (Size = Count) then begin
            { Формируем описание очередного образа }
        Description.ImageLeft := ImageDescription.ImageLeft;
        Description.ImageHeight:= ImageDescription.ImageTop;
        Description.ImageWidth := ImageDescription.ImageWidth;
        Description.ImageHeight:= ImageDescription.ImageHeight;
        D := ImageDescription.Flag and $07;
            { Выделяем три младших бита }
        D := 1 shl (D+1);  { 2 в степени (D+1) }
        Description.ImageColors := D;
        HasLocalColorMap := (ImageDescription.Flag and $80) < > 0;
            { Проверка установки старшего бита }
        Description.Interlace:= (ImageDescription.Flag and $40) > 0;
            { Флажок чередования }
        if HasLocalColorMap then begin
                { Чтение цветовой таблицы образа }
            Size := Description.ImageColors * sizeof(RGB) * 3;

            {$I-}
            BlockRead(F,Description.LocalColorMap,Size,Count);
            {$I+}

            If (IoResult < > 0) or (Size < > Count) then begin
                GetScreenImageDescription := IncorrectImageDescription;
                exit;
            end;
        end
        else begin
            for I := 0 to Description.Colors-1 do
            for J := Red to Blue do
                Description.LocalColorMap[I,J]:=
                    Description.GlobalColorMap[I,J];
            Description.ImageColors:= Description.Colors;
            Description.ImageColors:= Description.Colors;
        end
    end;
end;

Function GetRasterDataAndDisplay:Byte;
    { Считывание, дешифровка растровых данных, вывод на дисплей }
var
    W       :Word;
    x,xx    :Word;
    IoError :Integer;
    yy,y    :Word;
begin
    GetRasterDataAndDisplay := 0;
    InitBitsPerPixel := ReadByte(IoError);
    { Начальное число битов, используемое для кодирования пиксела }
    SetPaletteFromGif; { Установка палитры }
    InitDecoding;      { Инициализация данных }

    for yy := Description.ImageTop to Description.ImageTop +
              Description.ImageHeight - 1 do begin
        y := InterLace(yy);
    { Вычисление номера строки в режиме чередования }
        for x:= Description.ImageLeft to Description.ImageLeft +
                Description.ImageWidth - 1 do begin
            repeat
                W := GetPixelFromGif(IoError);
                if IoError > 0 then begin
                        { Обработка ошибки чтения данных }
                    GetRasterDataAndDisplay :=
                        ImageDataDecriptionError;
                        Exit;
                end;
                if W = CC then
                    InitCodeTable
                        { Инициализация таблицы LZW кодов }
                else
                if W = EOI then Exit
                        { Конец растровой информации }
                else PutPixelToScreen(x,y,W);
                        { Вывод очередного пиксела }
            until W < > CC;
        end; {for x}
    end; { for yy}
end;

Function GifInitGraph:word;
begin
    GrDriver := Detect;
    DetectGraph(GrDriver,GrMode);
    InitGraph(GrDriver,GrMode,'');
    if GraphResult = grOk then begin



        GifInitGraph := Ok;
        SetBkColor(Description.BackGround);
        SetFillStyle(SolidFill, Description.BackGround );
        Bar(0,0,GetMaxX,GetMaxY); { Закраска экрана цветом фона }
        GetPalette(SavedPalette);
                { Сохранение палитры для EGA, VGA 16 цветов}
    end
    else GifInitGraph := GraphInitError;
end;
```

Листинг 3 (продолжение)

```
Function GetRecordType:Byte;      {Чтение типа очередного блока }
begin
     GetRecordType := ReadByte(IoError);
end;

Function GetImageDescription:Byte;
                    { Чтение и проверка описания графического образа }

type
     ImageDsc = record
          ImageLeft, ImageTop          :integer;  { Левый верхний угол }
          ImageWidth, ImageHeight      :integer;  { Ширина, высота образа}
          Flag                         :Byte;     { Флажок }
     end;

var
     ImageDescription   :ImageDsc;
     Count, Size, D, i  :Word;
     j                  :RGB;
     HasLocalColorMap :Boolean;
begin
     GetImageDescription := Ok;

     {$I-}
     Size := SizeOf(ImageDescription);
     BlockRead(F,ImageDescription,Size,Count);
     If (IoResult < > 0) or (Size < > Count) then begin
          GetImageDescription := IncorrectImageDescription;
          Exit;
     end;
     {$I+}

     if (Size = Count) then begin
               { Формируем описание очередного образа }
          Description.ImageLeft := ImageDescription.ImageLeft;
          Description.ImageHeight: = ImageDescription.ImageTop;
          Description.ImageWidth := ImageDescription.ImageWidth;
          Description.ImageHeight: = ImageDescription.ImageHeight;
          D := ImageDescription.Flag and $07;
               { Выделяем три младших бита }
          D := 1 shl (D+1);   { 2 в степени (D+1) }
          Description.ImageColors := D;
          HasLocalColorMap := (ImageDescription.Flag and $80) < > 0;
               { Проверка установки старшего бита }

          Description.Interlace: = (ImageDescription.Flag and $40) > 0;
               { Флажок чередования }
          if HasLocalColorMap then begin
                    { Чтение цветовой таблицы образа }
               Size := Description.ImageColors * sizeof(RGB) * 3;

               {$I-}
               BlockRead(F,Description.LocalColorMap,Size,Count);
               {$I+}
               If (IoResult < > 0) or (Size < > Count) then begin
                    GetImageDescription := IncorrectImageDescription;
                    Exit;
               end;
          end
          else begin
               for i := 0 to Description.Colors-1 do
               for j := Red to Blue do
                    Description.LocalColorMap[i,j]: =
                         Description.GlobalColorMap[i,j];
               Description.ImageColors: = Description.Colors;
               Description.ImageColors: = Description.Colors;
          end
     end;
end;

Procedure TerminateGif;
var
     c:char;
begin
     SetAllPalette(SavedPalette);
               { Восстановление палитры для EGA, VGA, 16 цветов}
     CloseGraph;
end;
```

■

Листинг 4. Процедуры обслуживания вывода картинки

```
{Файл listing4.pas}

Function ReadByte(var IoError:Integer): Byte;     { Чтение байта данных }
var
     B         :Byte;
     Result    :Integer;
begin
     {$I-}
     IoError := -1;
     blockRead(F,B,1,Result);
     IoError := IoResult;
     ReadByte := B;
     {$I+}
end;

Procedure ReadByteBlock(Counter:Byte; var Buf:ByteBuffer;
                        var IoError:Integer);
                                   { Чтение в буфер Counter байтов }
var
     Result    :Integer;
begin
     {$I-}
     Ioerror := -1;
     if Counter > 0 then begin
          blockRead(F,Buf,Counter,Result);
          if (IoResult = 0 ) and ( Counter = Result) then IoError := 0;
          If (IoResult < > 0) then IoError := IoResult;
     end;
     {$I+}
end;

Function Interlace(y:word):Word;
                              { Реализация алгоритма чередования строк }
var
     yy, S:Word;
begin
     if Description.Interlace then begin
          yy := y - Description.ImageTop;
          S := Description.ImageHeight div 8;
          if (Description.ImageHeight mod 8) < > 0 then Inc(S);
          if yy < S then
                    { Первый проход: выводим каждую восьмую строку,
                    начиная с нулевой }
               Interlace: = Description.ImageTop + yy * 8
          else if yy < S * 2 then
                    { Второй проход: выводим каждую восьмую строку,
                    начиная с четвертой }
               Interlace: = Description.ImageTop + (yy-S) * 8 + 4
          else if yy < S * 4 then
                    { Третий проход: выводим каждую четвертую строку,
                    начиная со второй }
               Interlace := Description.ImageTop + (yy- S * 2) * 4 + 2
          else      { Четвертый проход: выводим каждую вторую строку,
                    начиная с первой }
               Interlace: = Description.ImageTop + (yy- S * 4) * 2 + 1 ;
     end
     else Interlace: = y;
end;

Procedure SetPaletteFromGif;              { Установка палитры }
const
     RedGreenBlue : array [RGB] of Byte = ($4,$2,$1);

var
     i         :word;
     j         :RGB;
     C,W       : word;
     R,G,B     :Byte;
     Color     :ShortInt;
begin
     if Description.Colors > = 16 then C: = 15
     else C := Description.Colors-1;
     for i: = 0 to C do begin
          Color: = 0;
          for j: = Red to Blue do begin
               W: = Description.LocalColorMap[i,j];
               W: = W shr 6;
               if (W and 2) > 0 then Color: = Color or RedGreenBlue[j];
               if (W and 1) > 0 then Color: = Color or (RedGreenBlue[j] shl 3);
          end;
          SetPalette(i,Color);
     end;
end;

Procedure PutPixelToScreen(x,y,W:word);   { Вывод пиксела на экран }
begin
     if (x < = GetMaxX) and (y < = GetMaxY) then
          PutPixel(x,y,W);     { Вывод на экран средствами BGI}
end;
```

■

Листинг 5. Декодирование растровых данных

```
{Файл listing5.pas}

const
     LzwTableSize  = 4095;          { Размер кодовой таблицы LZW}
     NonInited     = -1;            { Кодовая таблица не инициализирована }
```

```
    BitMask         :array[0..16] of Word =        { Битовая маска }
                            ($0000,$0001,$0003,$0007,
                            $000F,$001F,$003F,$007F,
                            $00FF,$01FF,$03FF,$07FF,
                            $0FFF,$1FFF,$3FFF,$7FFF,
                            $FFFF);

type
    CodeTableEntry = record      { Вход таблицы LZW кодов: }
        Code      :Word;         { Последний код в цепочке }
        Previous  :Integer;
                        { Ссылка на предыдущий код в таблице кодов }
    end;
    CodeTable      = array[0..LzwTableSize] of CodeTableEntry;
                                        { Таблица LZW кодов }
    Buffer         = array[1..255] of Byte; { Буфер данных }

var
    CodeBuffer                :ByteBuffer;
        { Буфер для упакованных LZW кодов}
    CurrentBitsPerPixel       :Byte;
        { Текущее число битов, используемое для кодирования пиксела }
    Table                     :CodeTable;
        { Таблица LZW кодов }
    PixelBuffer               :array    [1..LzwTableSize+1] of Byte;
        { Цепочка пикселов при декодировании}
    PixelsInBuffer            :Word;
        { Текущее число пикселов при декодировании}
    HiBit                     :Word;
        { Положение старшего бита при декомпрессии кодов }
    RootSize                  :Word;
        { Число основных кодов в LZW таблице }
    CurrentTableSize          :Word;
        { Текущий размер таблицы }
    CurrentCode,
    OldCode                   :Integer;
    W                         :Word;
    BitBuffer                 :LongInt;
        { Битовый буфер }
    ByteCounter               :Word;
        { Счетчик }
    ByteBufferSize            :Byte;
        { Текущий размер буфера }
    x,y,yy                    :Word;
        { Положение текущего пиксела }

Function GetByte:Byte;   { Чтение очередного байта данных }
begin
    if ByteCounter > ByteBufferSize then begin
        ByteBufferSize:= ReadByte(IoError);
        if IoError = 0 then
            ReadByteBlock(ByteBufferSize, CodeBuffer, IoError);
        ByteCounter:=1;
    end;
    GetByte:= CodeBuffer[ByteCounter];
    Inc(ByteCounter);
end;

Procedure NewTableEntry(TableIndex,ChainPos:Word);
                        { Формирование нового входа в LZW таблице }
var
    i:integer;
begin
    i:=TableIndex;
    PixelsInBuffer:=ChainPos;
    repeat                { Формирование потока пикселов }
        Inc(PixelsInBuffer);
        PixelBuffer[PixelsInBuffer] := Table[i].Code;
        i := Table[i].Previous;
    until i<0;

    Table[CurrentTableSize].Code := PixelBuffer[PixelsInBuffer];
    Table[CurrentTableSize].Previous := OldCode;
    Inc(CurrentTableSize);
    case CurrentTableSize of
        $2, $4, $8, $10,
        $20, $40, $80, $100,
        $200, $400, $800:
        Inc(CurrentBitsPerPixel);
            { Увеличение числа битов для кодирования пиксела }
        $1000:;
        else;
    end;
end;

Function GetCode:Word;       { Очередной LZW код }
var
    W:LongInt;
    B:Byte;
begin
    while HiBit < CurrentBitsPerPixel do begin   { Пополнение буфера }
        W:= GetByte;
        W:= W shl HiBit;
        Inc(HiBit,8);
        BitBuffer:= BitBuffer or W;
    end;
    W:= BitBuffer and BitMask[CurrentBitsPerPixel];
    BitBuffer:= BitBuffer shr CurrentBitsPerPixel;
    Dec(HiBit,CurrentBitsPerPixel);
    BitBuffer:= BitBuffer and BitMask[HiBit];
    GetCode:= W;
end;

Function GetPixelFromGif(var IoError:Integer) : Integer;
                { Извлечение очередного пиксела }
var
    W:Word;
begin
    if PixelsInBuffer = 0 then begin     { В буфере нет цепочки }
        CurrentCode:= GetCode;
        if (CurrentCode = CC) or
        (CurrentCode = EOI) then     { Особый случай }
            GetPixelFromGif:= CurrentCode
        else if OldCode=NonInited then begin
            GetPixelFromGif:= CurrentCode;
            OldCode:= CurrentCode;
        end
        else if CurrentCode < CurrentTableSize
        then begin            { Очередной код присутствует в таблице }
            NewTableEntry(CurrentCode,0);
            GetPixelFromGif:= PixelBuffer[PixelsInBuffer];
            Dec(PixelsInBuffer);
            OldCode:=CurrentCode;
        end
        else begin        { Кода нет в LZW таблице }
            NewTableEntry(OldCode,1);
            PixelBuffer[1]:=
            PixelBuffer[PixelsInBuffer];
                { Последний пиксел в цепочке равен первому }
            GetPixelFromGif:= PixelBuffer[PixelsInBuffer];
            Dec(PixelsInBuffer);
            OldCode:= CurrentCode;
        end
    end else begin  { Цепочка не пуста: извлечение пиксела из цепочки }
        GetPixelFromGif:= PixelBuffer[PixelsInBuffer];
        Dec(PixelsInBuffer);
    end;
end;

Procedure InitDecoding;        { Инициализация }
var
    i:word;
begin
    RootSize:= Description.Colors + 1;
    CC:= Description.Colors;      { Код очистки LZW таблицы}
    EOI:= RootSize;               { Код окончания данных}
    for i:=0 to RootSize do begin  { Инициализация LZW таблицы }
        Table[i].Code:=i;
        Table[i].Previous:=-1;
    end;
    InitCodeTable;
    BitBuffer:=0;
    HiBit:= 0;
    ByteCounter:=0;
    PixelsInBuffer:=0;
    ByteCounter:=1;
    ByteBufferSize:=0;
    IoError:= 0;
end;

Procedure InitCodeTable;             { Инициализация таблицы LZW кодов}
begin
    CurrentTableSize:= RootSize + 1;
    CurrentBitsPerPixel:= InitBitsPerPixel + 1;
    OldCode:=NonInited;
end;
```

Листинг 6. Контрольный пример

```
Program ShowGif;

uses gif, Crt;
var
    Error     : Byte;
    GifName   : String;
begin
    GifName:=ParamStr(1);      { Имя GIF файла из командной строки }
    Error:=GifViewer(GifName);
    Halt(Error);
                { Завершение в соответствии с кодом ошибки }
end.
```

Применение языка Ада для разработки программного обеспечения ПЭВМ

С. Позднеев

Исследования возможностей ПЭВМ показывают, что они достаточно эффективны при решении разнообразных задач обработки информации в таких сферах, как вычислительная и прикладная математика, разнообразные системы управления экспериментом в различных областях науки и техники и банковские и офисные системы. Основное достоинство ПЭВМ — это практически неограниченные возможности реализации разнообразных схем (видов) диалога [1], который позволяет исследовать достаточно большое число вариантов решения конкретной задачи и таким образом определить оптимальный вариант решения.

Эффективность обработки информации на ПЭВМ определяется следующими факторами [1], [2]:

- методами и средствами обработки информации и положенными в их основу математическими моделями;
- архитектурой ПЭВМ и соответствующего программного обеспечения (ПО);
- техническими особенностями элементной базы ПЭВМ, то есть технологией производства интегральных схем (ИС).

В настоящее время степень интеграции позволяет в пределах одной ИС реализовывать достаточно сложные устройства, такие как модули памяти, процессоры, интерфейсы и тому подобное. На основе ИС освоен выпуск отечественных серий ПЭВМ, а также большой набор средств расширения вычислительных возможностей ПЭВМ, включающий векторные и матричные процессоры. В результате этого появляется возможность создания высокоэффективных вычислительных систем (ВС) обработки информации, основанных на следующих принципах [1]-[5]:

- параллелизма процессов обработки информации;
- децентрализации средств управления и данных;
- программируемости структуры;
- модульности аппаратных и программных средств;
- асинхронности взаимодействия модулей и процессов;
- способности программно-технической системы к развитию.

Для обеспечения максимальной производительности подобной ВС необходимо решить задачу отображения алгоритмов обработки на архитектуру ВС, построенную на основе указанных принципов. Основой решения этой задачи является выбор языковых средств, реализующих это отображение.

Поэтому исследование и выбор языков программирования и основанных на них методов и средств построения ПЭВМ для оптимального решения конкретных задач представляет собой актуальную задачу.

В настоящее время для разработки ПО ПЭВМ все большее распространение получают языки высокого уровня (ЯВУ) BASIC, C, Pascal, Fortran, Modula, Ада и так далее. ЯВУ обладают целым рядом преимуществ перед языками низкого уровня, такими как простота создания, наглядность, удобочитаемость, модифицируемость и так далее.

Разработчику ПО ПЭВМ доступно, как правило, несколько ЯВУ и различных вариантов трансляторов. Например, для языка C и ОС типа MS-DOS на ПЭВМ типа IBM PC существует более 20 различных компиляторов, для языка Fortran и Pascal более пяти [5] и так далее.

Таким образом, задача выбора языка программирования, который соответствует принципам строгой типизации данных и конструкциям структурного программирования, обеспечивает мультизадачную и многопользовательскую организацию ПО с упрощенным программированием операций ввода-вывода и соответствует единой технологии программирования, является в настоящее время важной задачей. Следует отметить, что ЯВУ, такие как Algol, Fortran, COBOL не соответствуют всей совокупности указанных выше принципов. Одним из языков, соответствующих этим принципам, является язык Ада, который представляет собой новое поколение языков программирования [3]-[10]. Этот язык включает в себя все положительные стороны классических языков программирования, таких как Algol, COBOL, Fortran, PL/1 и обладает целым рядом новых свойств, обеспечивающих единую технологию процесса проектирования и разработки ПО. Здесь следует отметить особо, что только в рамках языка Ада проектирование, разработку (реализацию), отладку, эксплуатацию и модернизацию ПО можно представить как единый процесс.

Для этого в языке существуют следующие средства [3]-[9].

Строгая типизация данных, означающая, что типы всех значений данных, применяемых в ПО, контролируются на согласование и использование. Введение типов и их контроль являются чрезвычайно эффективными средствами для выявления ошибок в ПО. Система типов в языке Ада характеризуется:

- предопределенными типами, для которых определено множество значений и применяемых к ним операций;
- производными типами, которые определяются через ранее описанные типы;
- уточнениями, которые конкретизируют множество значений некоторого типа, но не влияют на количество применяемых операций;
- подтипами, учитывающими ограничения на исходный (родительский) тип, но относящиеся к тому же типу, что и исходный;
- запросами атрибутов, которые позволяют программисту узнавать атрибуты типов.

Тип определяет не только диапазон возможных значений, но и способ представления данных, а также набор операций, допустимых для элементов данных этого типа.

Смысл строгой типизации означает, что программистам следует поддерживать отличие разных элементов данных, которые они применяют. Производные типы наследуют свои свойства от уже существующих типов, так что для нового типа сразу существуют аналогичные значения, и операции, применяемые к таким значениям, являются теми же самыми, что и для исходного типа.

Абстрактные типы данных имеют более общий характер в том смысле, что новый тип имеет явно определенный набор свойств, и программист задает полный набор операций, допустимый для значений этого типа.

Достаточно часто возникает необходимость использования общих данных, часто выполняемых процедур и так далее. Эти структуры данных или набор операций можно объединить в виде пакета, что позволяет исключить повторное их описание в каждой части ПО, ссылающейся на эти данные. В дальнейшем из любого места ПО (при условии, что описание этого пакета в данном месте удовлетворяет определенным условиям) можно делать ссылки на объекты пакета.

При разработке ПО достаточно часто возникает случай, когда необходимо запрограммировать действия со своим специальным типом таким образом, чтобы подпрограммы работали со значением этого типа и существовали средства, запрещающие применение других операций к объектам этого типа. Это возникает при разработке блоков управления, специализированных интерфейсов и так далее. Тогда эти части ПО должен описать разработчик блока. Такие типы данных называются абстрактными. При этом спецификация пакета разбивается на две части — видимую и личную, в последнем случае пользователю пакета доступна только та информация, которая ему необходима.

Строгое описание любых объектов на абстрактном уровне характеризуется прежде всего набором применяемых к ним операций, а не деталями описания их внутренней структуры. Подобная типизация обеспечивает наглядность, что позволяет выявить основное число ошибок во время трансляции. В языке Ада абстрактный тип данных реализуется при помощи понятия пакета и предоставляемых им типов и процедур. Для объектов указываются тип и процедуры, в параметрах которых встречаются объекты этого типа. Чтобы использовать этот абстрактный тип данных, достаточно описать объект предоставляемого пакетом типа и использовать его при вызове процедур. Возможность настройки абстрактного типа данных реализуется в языке Ада либо с помощью настройки типа (например, указание размера массивов, входящих в его описание), либо созданием настраиваемого пакета (generic package), в этом случае при помощи указания параметров настройки могут быть созданы новые пакеты.

Обработка исключительных ситуаций (взаимных блокировок различных ресурсов ПЭВМ).

Организация взаимодействия задач внутри ПО средствами самого языка (метод рандеву), не используя средства ОС. ЭТОТ метод обеспечивает связь и синхронизацию параллельно выполняемых задач.

Механизм вложения и раздельной компиляции. Основная единица программ — единица компиляции. Каждый модуль или пакет включает две единицы компиляции: описание пакета и тело пакета. Описание (спецификация) пакета — это список всех элементов типов данных, подпрограмм, функций, задач, исключительных ситуаций и так далее, который делится на видимую и приватную части. Элементы, описанные в видимой части спецификации пакета, доступны другим программным единицам. Информация, содержащаяся в приватной части спецификации, доступна компилятору, но не доступна программисту при написании других компонентов. Таким образом, упрятывание какой-либо информации в приватную часть спецификации уменьшает взаимосвязь различных программных компонентов и упрощает последующее внесение изменений в ПО. Ограничение видимости или доступности отдельных пакетов также повышает модульность разрабатываемого ПО. Аналогичная ситуация возникает и при разработке

модулей. Модули в языке Ада могут использоваться в качестве поименованной совокупности описаний типов данных и переменных, для группировки логически связанных подпрограмм и для определения абстрактных типов данных и операций над ними, в этом случае для остальных программных компонентов внутреннее устройство данных и реализация операций над ними остаются невидимыми.

Исходный текст программы может быть представлен компилятору в виде единиц компиляции, которые образуются посредством выделения тела подпрограммы, тела пакета или тела задачи из раздела описаний родительской единицы в самостоятельно компилируемую единицу. Спецификация единицы (подъединицы) компиляции задается в разделе описаний родительской единицы, а на месте вынесенного тела подъединицы описывается специальная конструкция — след тела (body slab). Библиотечная единица — это единица компиляции, которая не является подъединицей никакой другой единицы компиляции.

В языке Ада предусмотрены следующие основные средства задания связей между единицами компиляции [3], [6], [7]:

- фразы контекста (with), описывающие имена библиотечных единиц, используемых в данной единице компиляции;
- указание имени родительской единицы в заголовке подъединицы;
- описание в родительской единице на месте вынесенного тела подъединицы специальной конструкции — следа тела;
- задание прагмы подстановки (in line) для некоторой подпрограммы, что означает необходимость прямой подстановки тела данной подпрограммы на месте каждого ее вызова.

Programmer

Язык Ада позволяет реализовать единую технологию программирования, соответствующую принципам строгой типизации данных, структурного программирования и обеспечивающую мультизадачную и многопользовательскую организацию ПО.

Эти средства обеспечивают точный учет и контроль связей между единицами компиляции при их семантическом анализе и используются в качестве указаний, определяющих необходимость и порядок компиляции (перекомпиляции) единиц и их хранения в библиотеке единиц компиляции.

В этой библиотеке хранятся результаты обработки единиц компиляции, которые могут быть использованы при работе с другими единицами. Поэтому средство раздельного компилирования позволяет осуществить физическое структурирование ПО в соответствии с его логической структурой, которая позволяет разбивать ПО на языке Ада на автономные подсистемы (процедуры, функции, пакеты, задачи). Это означает, что одна или несколько единиц ПО могут компилироваться отдельно, а затем объединяться с другими программными единицами, оформленными в виде сегментов компиляции.

Применение физического структурирования ПО позволяет на практике реализовать оба современных стиля разработки программ: сверху вниз и снизу вверх. Следует отметить, что логическое и физическое структурирование ПО имеет важную роль не только на этапе проектирования, но также на всех этапах создания ПО.

Здесь следует отметить особо, что благодаря перечисленным особенностям, в отличие от общепринятого метода разработки ПО, основанного на управлении, язык Ада поддерживает объектно-ориентированное программирование (ООП) [6]-[9]. Этот стиль программирования характеризуется тем, что текст программы описывает создание и взаимосвязь независимых и строго определенных структур данных, называемых объектами. В методе, основанном на управлении, программа рассматривается как управляющая последовательность действий над множеством ее структур данных, причем основной недостаток этого стиля программирования состоит в том, что по мере возрастания сложности ПО становится практически невозможно сохранить полное представление обо всех последовательностях выполняемых программой действий.

В методологии ООП программирование начинается со связывания одной-единственной структуры данных с фиксированным набором программ, причем единственными операциями, определенными над этими объектами, являются соответствующие действия.

Здесь необходимо отметить, что, в отличие от общепринятых метрических характеристик программ, для количественной оценки ПО, разрабатываемого на языке Ада, наиболее удобны следующие показатели [10], [11]:

- количество описанных, настраиваемых или исходных пакетов;
- количество конкретизаций или вхождений настраиваемых пакетов;
- частота использования нелокальных компонентов или объектов, которая может служить мерой сложности модификации ПО при внесении изменений;
- видимость пакетов;
- возможность осуществления машинно-зависимого программирования для работы со специализированными устройствами. Для этого в языке Ада выделен уровень спецификации представления, который позволяет устанавливать соответствие между логическими и физическими описаниями. Это дает возможность включать в прикладные программы на языке Ада подпрограммы, разработанные на языке ассемблера и в машинном коде соответствующих ЭВМ, и включать процедуры, упрощающие обработку прерываний от внешних устройств;
- средства, облегчающие программирование ввода-вывода файлов, состоящих из данных одного типа и текстов. Помимо этого, пользователю предоставлена возможность создавать свои собственные программные единицы ввода-вывода, специфичные для данной предметной области;

- средства, предоставляющие пользователю возможность обработки исключительных ситуаций;
- средства тестирования и отладки исходного текста программ и средства интеграции в единую систему ПО [7]-[9], [12], [13]. В этой системе поддержки процесса программирования в языке Ада APSE (Ada Programming Support Enviroment) выделяются следующие функциональные уровни: KAPSE (Kernel APSE) — ядро системы, MAPSE (Minimal APSE) — минимальный набор инструментальных средств и APSE — система, получаемая из MAPSE усилиями пользователей.

В состав KAPSE входят:

- операционная система реального времени, предназначенная для поддержки разработки и отладки программ на языке Ада;
- база данных, предназначенная для хранения отлаживаемых программ и данных.

В состав MAPSE входят:

- трансляторы Ада;
- редактор текста;
- отладчики.

В состав APSE включают (например, фирма ARTEK [6]-[8]):

- полноэкранный редактор с макроязыком, указывающий место и тип ошибок;
- редактор связей-библиотекарь;
- эмулятор-отладчик, который выполняет программу в промежуточных А-кодах, эмулируя работу виртуальной машины, он позволяет анализировать содержимое стеков, регистров и так далее. Работа с А-кодом дает возможность достигать более высоких результатов при отладке, по сравнению с традиционными отладчиками в символических кодах;
- дисассемблер;
- оптимизатор.

В состав APSE могут также входить средства автоматизированного специфицирования программ и создания документации на разрабатываемое ПО.

Основными направлениями в разработке средств поддержки программирования на языке Ада в настоящее время являются:

- создание более дешевых компиляторов, не поддерживающих всех соглашений языка Ада;
- создание компиляторов с более мощными средствами оптимизации (в частности, позволяющими получать более эффективные и компактные исполняемые коды);
- создание с помощью языка Ада системных средств, позволяющих более эффективно поддерживать реализацию прикладных программ на языке Ада (принцип языкового единства). Например, на языке Ада разработаны трансляторы с языка Ада, операционная система распределенной обработки информации PULSE (подобная системе UNIX), СУБД типа INGRES и специализированный язык AdaPlex, при помощи которого реализованы СУБД DDM, MULTIBASE и так далее [14], [15].

Язык AdaPlex предназначен для решения задач, связанных с применением баз данных [14]. Этот язык построен на основе введения в язык Ада языка управления данными DataPlex [14]. Язык AdaPlex реализует более широкие возможности моделирования предметной области, чем иерархическая, сетевая или реляционная модель данных, то есть относится к так называемым функциональным моделям данных и поддерживает, например, понятие сущности, функции и обобщенной иерархии. Сущности представляют собой концептуальные объекты. Сущности, имеющие множество свойств, объединяются в множества сущностей или типы сущностей. Функции соответствуют свойствам объектов. Применение функции к сущности дает в результате одно или множество значений. Обобщенная иерархия означает специальное отношение между множествами сущностей, когда одно множество сущностей, называемое подтипом, имеет все свойства (функции) другого множества сущностей, называемого супертипом. В языке AdaPlex существует новая форма выражений, в отличие от конструкций языка Ада, характерной особенностью которых является то, что в них определяются условия, в соответствии с которыми сущности извлекаются из баз данных. В этом языке введены новые операторы для включения сущности в тип

```
include выражение in type имя типа
```

или исключения сущности из типа

```
exlude выражение from type имя типа
```

Введена новая форма оператора цикла, предназначенная для последовательной обработки всех элементов множества

```
for each параметр цикла in домен
```

В этом операторе при каждом выполнении цикла параметр получает значение очередного значения домена. Модуль в языке AdaPlex представляет специальную форму пакета. По своей природе модули разделяются на два типа: модули базы данных и модули вида. Модули базы данных представляют собой совокупность описаний типов сущностей (схем модулей), деклараций целостности и подпрограммы. Для некоторых целей модули базы данных могут быть преобразованы в модули вида. Основная особенность модулей состоит в том, что они могут одновременно функционировать с несколькими основными программами. Подобно пакету, модуль состоит из двух частей — спецификации модуля и тела модуля. Все данные, принадлежащие модулю, порождаются соответствующим модулем базы данных, что и описывается в приватной части. Благодаря этим свойствам язык AdaPlex интенсивно применялся для создания распределенных однородных и неоднородных систем баз данных. Технологические средства поддержки на языке Ада — это программа STAIRS.

Благодаря системе поддержки, а также специальным конверторам типа DIANA, ANNA [6], [7] с других языков в настоящее время уже переведены на язык Ада наиболее универсальные библиотеки программ NAG,MSLIB и так далее [3]-[8].

Описанные свойства языка Ада определили его широкое применение для решения задач вычислительной математики, системного программирования, разработки систем распределенной и параллельной обработки информации и тому подобных.

Сложность задачи выбора оптимального компилятора с языка Ада на ПЭВМ связана с множеством противоречивых требований, таких как скорость трансляции, скорость выполнения программ, соответствие стандарту, наличие системы поддержки, возможность компиляции и генерации ПО на различных ЭВМ. В табл. 1 приведены характеристики наиболее распространенных компиляторов с языка Ада [7]-[9], что, по нашему мнению, предоставляет определенные удобства для выбора компилятора.

В этой таблице особо следует отметить компиляторы второго поколения, например, Telegen 2 фирмы Telesoft, который состоит из
- непосредственно компилятора;
- препроцессора;
- библиотеки программ;
- системы управления библиотекой;
- генератора объектного хода;
- оптимизатора.

Таблица 1

Компиляторы и системы программирования на языке Ада

Наименование компилятора или средств программирования	Фирма-разработчик	Программно-техническая среда			Дополнительные возможности	Примечания
		Тип ЭВМ	Операционная система	Дополнительные требования		
ALSYS Ada	ALSYS Inc.	IBM PC/XT, AT и совместимые с ними, VAX, IBM 370, HP 1000A, HP 9000	MS-DOS, UNIX, VAX/VMS, MVS		Компиляторы Alsys позволяют использовать всю адресуемую память ЭВМ без ограничений, которые накладываются ОС	Для компиляторов Alsys разработано системное окружение типа APSE и имеется полный комплект документации
ALSYS Compiler Plus	ALSYS Inc.	IBM PC/AT и совместимые с ними	MS-DOS		Дополнительно поставляется обучающий пакет Ada QUERY	
PC VADS	Verdix Corp.	IBM PC/XT, AT и совместимые с ними	MS-DOS или UNIX 2.0		Система программирования VADS (Verdix Ada Development System)	
Artek Ada	Artek Corp.	IBM PC/XT, AT и совместимые с ними	MS-DOS, OS-2	Специализированный процессор для оптимизации А-кода	Работа с А-кодом	Имеются полная и ограниченная версии языка Ada без параллельного программирования.
CS/ICC	General System	IBM PC/XT, AT и совместимые с ними	MS-DOS	Дополнительный процессор 8087		Не прошел процедуры аттестации
Ada/ED-C Interpreter	New York University	IBM PC/XT, AT и совместимые с ними	MS-DOS, Concurrent DOS 286	Для MS-DOS, Concurrent DOS		Сравним с интерпретатором BASIC. Используется в основном для обучения
JANUS Ada, C-PAK, D-PAK, S-PAK	RR Software	IBM PC/XT, AT и совместимые с ними Zenith Z1008, IBM 370, HP 1000A, HP 9000	MS-DOS	Для Zenith, процессор Z 80		Ограниченная версия. Отсутствуют динамические множества, настраиваемые пакеты и методы параллельного программирования. Применяется для обучения

Вследствие сложности и емкости языка Ада обычно возникают опасения, что разработанное на этом языке ПО будет требовать значительных временных ресурсов, по сравнению с ПО, реализованным на традиционных языках C, BASIC, Fortran, Pascal и языках четвертого поколения, то есть непроцедурных (QBE, QBF, SQL и с применением пакетов параметрического программирования типа NATURAL [13]-[15]).

С этой целью была проведена сравнительная оценка эффективности следующих трансляторов:
- языка BASIC в режиме интерпретации;
- языка BASIC в режиме компиляции;
- языка Ада в режиме компиляции;
- языка C (компилятор Innovation);
- языка C (компилятор MS, режим без оптимизации);

Таблица 1

Компиляторы и системы программирования на языке Ада

DDS Ada Compiler	Advanced Computer Techniques	VAX, IBM PC, iAPX 286	VAX/VMS, MS-DOS 1790A			
DEC Ada	DEC	VAX, PDP	VAX/VMS			
R100 Development System	Rational	IBM 370, VM/CMS, MVS/TSO			Поддержка всего жизненного цикла проектирования, разработки, отладки, сопровождения и модификации программного обеспечения	
AN/UYK-44	Softech	IBM 370	VM/CMS, MVS/TSO, UNIX, Beam, NTT-Dips			
Telesoft Ada Compiler Telegen	Telesoft	VAX, IBM 370, PDP, Appolo, IBM PC/XT, AT, Symbolics Could	VAX/VMS, MVS, VRTX, MS-DOS, OS-2		Для этих компиляторов существуют оптимизаторы, системы динамического анализа, специальные преобразователи кода	Самые производительные компиляторы Ada
IBM Ada Compiler	IBM	IBM 4381	MVS 3.8			
Intermetrics Inc., 370/CMS Ada Compiler V 201.16	Intermetrics Inc.	IBM 3083	VM/CMS, MVS/TSO			
Intermetrics Inc., 370/UTS Ada Compiler	Intermetrics Inc.	IBM 3083, IBM 4341	UTS 2.3			
Intermetrics Ada Integrat. Environment (AIE)	Intermetrics Inc.	Все ЭВМ марки IBM	Все ОС, поддерживаемые фирмой IBM			Средство разработки ПО
IBM PDL/Ada (Program Design Lanquage/Ada)	IBM	Все ЭВМ марки IBM	Все ОС, поддерживаемые фирмой IBM			Средство разработки ПО
Intermetrics Byron/Ada Tools Set	Intermetrics Inc.	Все ЭВМ марки IBM	Все ОС, поддерживаемые фирмой IBM			Средство разработки ПО
Ada Compiler Validation Capability AVC						Система аттестации Ada компиляторов
Amdahl Ada Compiler	Amdahl	Amdahl 470/7, A, B, C; Amdahl 480/8; Amdahl 580				

- языка С (компилятор MS, режим максимальной оптимизации);
- языка С (компилятор MS, режим оптимизации по времени исполнения программы);
- языка С (компилятор MS, режим оптимизации для оперативной памяти);
- языка KMAN (режим интерпретации).

Для получения сравнимых результатов была выбрана программа, реализующая алгоритм подсчета простых чисел по методу решета Эратосфена [7]-[9]. Реализация данного алгоритма на выбранных языках компактна по объему исходного текста (не более 3 Кбайтов, 12 исполняемых операторов, организующих содержательную обработку), однородна для указанных языков по типу исполняемых операторов и типу данных. Используемый алгоритм позволяет оценить работу сравниваемых языковых средств в области реализации часто встречающихся программных конструкций типа "while", "if then", "for", обработки массивов натуральных чисел и логических переменных. Следует отметить особо, что поскольку рассматриваемые языки поддерживают указанные конструкции и типы данных, то реализуемые программы аналогичны по структуре и не используют специфических особенностей языков.

Соответствующее указанному алгоритму ПО поставляется, как правило, в качестве демонстрационного примера для проверки правильности установки транслятора. Эксперимент проведен в операционной системе MS-DOS на ПЭВМ типа IBM PC (ЕС-1845). Выборка производилась на множестве из 1899 простых чисел.

Результаты измерений представлены в табл. 2, 3. Следует отметить, что измерения проводились с целью получения сравнительных характеристик. Полученные результаты показывают линейную зависимость времени выполнения программы от числа итераций для перечисленных языков. При увеличении числа итераций относительное увеличение времени остается в целом постоянным (например, для языка С при компиляции без оптимизации, по сравнению с максимальной оптимизацией, увеличение времени составляет 33%), однако абсолютное увеличение времени растет. Показатели относительного увеличения времени выполнения программ, по сравнению с максимально "быстрой" программой (язык С, максимальная оптимизация), приведены в табл. 2. Таким образом, для часто исполняемых программ, например, компонентов ОС или трансляторов, абсолютное увеличение времени может достичь значительных величин. Эти данные согласуются с аналогичными измерениями, проведенными для трансляторов Alsys Ada, Lattice C, Turbo Pascal, описанными в [8], [9].

На основании приведенного материала можно сделать следующие выводы.

1. Язык Ада обеспечивает средства создания единой технологии проектирования и реализации ПО при помощи абстракции данных и их строгой типизации, логической и физической структуризации ПО, принципа раздельного компилирования, описания и тел единиц (подъединиц) разрабатываемых программ, упрощения машинно-зависимого программирования и программирования исключительных ситуаций.

2. Имеющаяся на ПЭВМ реализация языка Ада [5]-[9] позволяет получить достаточно эффективное по времени выполнения ПО (табл. 1, 2), сравнимое по эффективности с широко распространенными языками программирования.

3. Наличие технологического окружения Ада компилятора APSE (KAPSE, MAPSE и APSE) типа VADS [3]-[12], реализованного на ПЭВМ типа IBM PC/XT, предоставляет пользователям системы программ, мобильность и надежность которых значительно выше, по сравнению с ПО, реализованным на других ЯВУ.

Таблица 2

Время выполнения (с) алгоритма [5] для различных языков и соответствующих трансляторов

№ п/п	Языки, трансляторы	Коэффициент расширения	Число итераций текста				
			10	100	500	1000	1500
1	BASIC интерпретатор	-	686	-	-	-	-
2	BASIC компилятор	4	129	1188	-	-	-
3	JANUS/Ada	8	5	48	240	479	718
4	C-Innovation	3	5	53	262	524	786
5	C-MS оптимизации нет	5	4	38	187	375	563
6	C-MS оптимизация max	5	3	28	140	281	422
7	C-MS оптимизация по времени	5	3	28	140	281	422
8	C-MS оптимизация по объему	5	3	28	140	281	422
9	Knowledge-manager	-	3650	-	-	-	-

4. Наличие стандарта и средств сертификации языка Ада и комментариев к ним, высокий уровень языка создают дополнительные удобства эксплуатации и модификации ПО на языке Ада, а существующие компиляторы языка вместе с обучающими пакетами позволяют использовать язык Ада для обучения пользователей различной квалификации.

5. Возможность построения на базе языка Ада языков более высокого уровня и открытость сверху позволяют использовать его в качестве базового для реализации различных языковых средств сложных систем управления.

6. Возможность реализации СУБД с использованием языка Ада (например, INGRES) позволяет использовать его в качестве языкового окружения систем баз данных.

7. Открытость языка снизу (промежуточный А-код) позволяет разрабатывать переносимые программы для различных типов специализированных ЭВМ. Системные возможности языка позволяют реализовать протоколы сетей, а решение задач в параллельном режиме позволяет эффективно организовать вычислительный процесс в системах распределенной обработки.

8. Перечисленные особенности обеспечивают решение задачи языкового единства в региональных и национальных системах обработки информации, включающих в себя неоднородные сети данных.

Технологическое окружение языка Ада позволяет эффективно организовать и поддерживать все этапы жизненного цикла ПО, что делает использование Ада перспективным при создании новых технических средств.

В заключение автор выражает искреннюю благодарность Аленичеву В.И. за реализацию основных расчетов, а также сотрудникам Минского центра по изучению систем программирования на основе языка Ада за участие в дискуссиях. ■

Литература

1. Брябрин В.М. Программное обеспечение персональных ЭВМ.– М.: Наука, 1989.
2. Kubler F.D. HighPerformance Computing.– Berlin: Parasytec, 1992.
3. Джехани Н. Язык Ада.– М.: Мир, 1988.
4. Василеску Ю. Прикладное программирование на языке Ада.– М.: Мир, 1990.
5. Бар Р. Язык Ада в проектировании систем.– М.: Мир, 1988.
6. Языки программирования Ада, Си, Паскаль. Сравнение и оценка/ Под ред. А.Р.Фьюэра, Н.Джехани.– М.: Радио и связь, 1989.
7. Янг С. Алгоритмические языки реального времени. Конструирование и разработка.– М.: Мир, 1985.
8. Dimitris Christodourakis. Ada: The Choice for '92.– N.Y.: Springer-Verlag, 1992.
9. Shammas N.S. Ada Moves to Micros// Byte.– 1987.
10. McGarry F.E., Agresti W.W. Measuring Ada for Software Development in the Software Engineering Laboratory// J. Syst. Software.– 1989.– Vol. 9.
11. Withrow C. Prediction and Control of Ada Software Defects// J. Syst. Software.– 1990.– Vol. 19.
12. Luckham D. Programming with Specification.– N.Y.: Springer, 1990.
13. Long F. Software Engeneering Enviroments.– N.Y.: Springer, 1990.
14. Чжань А., Дейл У., Фокс С. Системы управления распределенными базами данных, ориентированные на язык Ада// ТИИЭР.– 1987.– Т. 75.– № 5.
15. Chung C.W. DATAPLEX: An Access to Heterogeneous Distributed Databases// Comm. ACM.– 1990.– Vol. 33.– № 3.

Таблица 3

Результаты измерений времени (с) выполнения программ на алгоритмических языках С, Ада, Pascal

Алгоритм	Компиляторы				
	Alsys Ada с оптимизацией	Alsys Ada без оптимизации	Lattke C Revision 7.5	Lattke C Revision 7.5	Turbo Pascal
Все перемещения 7 целых чисел (5 раз рекурсивно)	3.34	2.31	2.86	2.41	4.56
Перемножение квадратных матриц размером 40*40	6.09	5.10	12.31	2.53	3.52
Перемножение двух матриц 32-разрядных чисел с плавающей точкой	11.09	9.87	40.48	29.28	79.58
Быстрое преобразование Фурье (256 точек FFT выполнено 20 раз)	13.46	11.85	61.29	52.78	121.33
Задача восьми ферзей (решена 50 раз)	2.75	1.81	4.11	1.31	2.96

От 2400 до 19200: не ругайте наши линии связи

Е. Равич

Достаточно открыть практически любую газету, от центральных до местных, как в глаза бросятся многочисленные рекламные объявления крупных и не очень крупных фирм, предлагающих самую разнообразную вычислительную технику и периферийные устройства, в том числе модемы и факс-модемы десятков производителей. На всех компьютерных выставках последних лет коммуникационное оборудование занимает почетное место и вызывает заслуженно большой интерес. Даже не проводя специального анализа рынка, можно, не сильно ошибившись, предположить, что вниманию потенциальных покупателей предлагается около двухсот импортируемых моделей, изготовленных в США, Западной Европе, Юго-Восточной Азии, причем в различных исполнениях. Как правило, это достаточно дешевые низкоскоростные изделия, часть из которых, по словам их продавцов, «адаптирована» к отечественным телефонным линиям или даже сертифицирована в нашем Министерстве связи. Возможно, это и так. Но по опыту, как собственному, так и многочисленных пользователей модемов России (а информация о действительно высококачественных изделиях расходится очень быстро), могу сказать, что далеко не все предлагаемые модемы оправдывают надежды их покупателей. Прежде всего, есть претензии к устойчивости работы модемов на линии, скорости, надежности, быстроте соединения, совместимости, качеству документации. В то же время на рынке модемов уже есть лидеры (явно обозначившиеся в настоящее время или потенциальные).

Для данного обзора были выбраны изделия, представляющие основные секторы рынка: низкоскоростные (2400 бит/с) модемы с аппаратной поддержкой протоколов сжатия данных и коррекции ошибок, факс-модемы (2400 бит/с — 9600 бит/с) с аналогичными характеристиками и высокоскоростные изделия (9600 бит/с и выше). Низкоскоростные модемы без аппаратной поддержки протоколов сжатия/коррекции быстро теряют популярность (как среди зарубежных, так и отечественных пользователей), а высокоскоростные становятся дешевле, совершеннее и, как следствие, все более распространенными. В первую очередь, интерес вызвали модели высокого качества, но доступные для массового отечественного потребителя. Производители рассматриваемых в обзоре изделий, фирмы Intel, Zoom и ZyXEL, всемирно известны и вряд ли нуждаются в дополнительном представлении. Достаточно сказать, что модемы Zoom являются одними из самых популярных на рынке США, а высокоскоростные модели ZyXEL серии U-1496 получили высшие оценки по результатам сравнительного тестирования модемов в многочисленных компьютерных журналах США и Западной Европы.

Все рассматриваемые модели в настоящее время доступны для потребителей России и ближнего зарубежья. Некоторые их характеристики приведены в таблице 1. Автор благодарит фирмы Intel и «Вариант», предоставившие модемы для тестирования. Факс-модем Zoom был приобретен у фирмы НАТЕКС.

Процедуры тестирования

Главной целью тестирования изделий было выяснение возможности для потенциальных покупателей, получивших стандартную поставку, успешно работать на отечественных линиях связи на объявленных поставщиками скоростях передачи данных и при необходимости использовать другое коммуникационное программное обеспе-

Таблица 1

МОНИТОР НАШ ВЫБОР

	ZyXEL 1496E Plus	Intel 2400e V.42bis	Zoom FX-9624V
Модем	+	+	+
Факс-модем	+	-	+
Факс/модем/речь	+	-	-
Номинальная скорость передачи данных, бит/с	см. примечание	2400	2400/9600
Реализация протоколов			
V.42/MNP4	+	+	+
V.42bis/MNP5	+	+	+
Используемые наборы микросхем	Motorola Texas Instr. ZyXEL	Intel	Rockwell
Простота установки и конфигурирования	+	+	+
Удобство работы, требования к качеству линии	+	+	+
Совместимость с другими модемами	+	+	+
Прилагаемое ПО	ZFAX другие утилиты	Communicator 2.0 TESTCOM	MTEZ 1.17 Express Fax

Примечание. Для модемов ZyXEL 1496E Plus предусмотрена возможность замены (перепрограммирования) ПЗУ модема при появлении новых версий управляющей программы (firmware). Это позволяет с минимальными затратами постоянно улучшать эксплуатационные характеристики модема, в частности, увеличить номинальную скорость передачи данных до 19200 бит/с и использовать новые версии высокоскоростного протокола фирмы ZyXEL. В начале 1993 года было объявлено о начале поставок управляющей программы версии 6.0.

чение. Обращалось внимание на комплектацию, качество и полноту прилагаемой документации и программных средств, простоту установки, удобство работы. Тестировались только те возможности изделий, которые характерны именно для модемов.

Для проверки совместимости изделий с другими модемами проводились сеансы связи с московскими BBS и некоторыми сетями передачи данных, причем с целью получения более точной информации о требованиях модема к качеству линии сеансы с каждым абонентом повторялись минимум 2 раза (в дневное и вечернее время).

Собственно тестирование происходило с использованием компьютера с достаточно часто встречающейся конфигурацией (Intel 80286, AMI BIOS, UART NS16450) и коммуникационной программы Telix 3.25 без использования внешних протоколов. Выбор программы определялся следующими соображениями:

- программа не входила в стандартную поставку ни одного изделия, поэтому несовместимость с каким-либо модемом выявилась бы достаточно быстро;
- программа достаточно распространена среди отечественных пользователей персональных компьютеров и модемов и заслуженно считается одной из лучших в своем классе (Telemate, Procomm Plus, Bitcom);
- программа поддерживает практически все распространенные протоколы передачи файлов, что позволит дополнительно определить пригодность некоторых из них при работе на линиях плохого качества и повышенных (до 19200 бит/с) скоростях.

В качестве контрольного использовался модем Viva 24im фирмы Computer Peripherals (США), с которым автор интенсивно работал в течение года. Модем представляет собой пример достаточно надежного изделия с аппаратной поддержкой протоколов MNP, находящегося в США в середине ценового диапазона и хорошо зарекомендовавшего себя в процессе эксплуатации на отечественных линиях связи.

Intel 2400e V.42bis

Комплектация поставки

Модем 2400e V.42bis, блок питания 120В/60Гц, телефонный кабель RJ-11, руководство пользователя модема(Intel 2400 V.42bis/ 2400e V.42bis User Manual), коммуникационная программа Crosstalk Communicator 2.0 и программа тестирования модема/последовательных портов TESTCOM, руководство пользователя Crosstalk Communicator.

Общие замечания

Модем 2400e V.42bis, младшая модель семейства модемов фирмы Intel, реализован на базе набора специализированных микросхем для конструирования модемов 89C024FT (Intel 89C024FT V.42/42bis modem chip set). Набор состоит из микросхемы сопряжения с линией 89027, микроконтроллера 89C026FT (на базе 80C196) и различных видов ПЗУ. Аппаратно реализуются протоколы коррекции ошибок (V.42 и MNP4) и протоколы сжатия данных (V.42bis и MNP5). Поддерживаются международные стандарты V.21, V.22, V.22 bis, а также стандарты BELL 103 и BELL 212A. Применение специализированных микросхем высокой степени интеграции позволило уменьшить размеры модема, снизить потребляемую мощность, повысить надежность передачи данных.

Программное обеспечение, поставляемое с модемом, отличается от наиболее часто встречающихся в нашей стране коммуникационных средств. Программа TESTCOM фирмы Intel позволяет быстро проверить работоспособность модема и последовательных портов и может быть оценена как очень полезная. Коммуникационная программа Crosstalk

Communicator 2.0 фирмы DCA (один из многочисленных представителей семейства программ Crosstalk) представляет собой усеченную версию гораздо более мощной программы Crosstalk Mk.4. Интерфейс Communicator несколько непривычен (по сравнению с интерфейсом более распространенных программ Telix, Telemate, Quick Link, Procomm Plus, MTEZ) и, на мой взгляд, может вызвать затруднения у неподготовленного пользователя. Возможности для создания сценариев (в Communicator 2.0 используется подмножество языка CASL) также невелики.

Блок питания (в предоставленном для тестирования комплекте) был рассчитан на напряжение 120В/60Гц. Можно надеяться, что у дистрибьютеров продукции фирмы Intel эта небольшая проблема будет решена.

Индикация в процессе работы модема

Набор индикаторов на передней панели модема несколько отличается по составу и расположению от наиболее распространенных. Некоторые индикаторы многофункциональны и меняют цвет в зависимости от особенностей конкретного сеанса передачи данных и в процессе сеанса. Индикаторы перечислены в соответствии с расположением на панели (слева направо).

V42 (V.42bis/V.42)	Зеленый цвет индикатора соответствует режиму V.42, желтый цвет — режиму V.42bis (в момент соединения). Красный цвет сигнализирует об ошибке в данных.
MNP (MNP)	Зеленый цвет индикатора соответствует режиму MNP 1-4, желтый цвет — режиму MNP 5 (в момент соединения). Красный цвет сигнализирует об ошибке в данных.
HS (High Speed)	Зеленый цвет соответствует скорости последовательного порта 2400 бит/с и выше (при использовании режима V.42bis скорость последовательного порта может быть установлена до 9600 бит/с).
AA (Auto Answer)	Зеленый цвет сигнализирует о том, что модем находится в состоянии автоответа.
CD/OH (Carrier Detect/ Off Hook)	Красный цвет индикатора соответствует отсутствию несущей, желтый — определению несущей и установлению связи между модемами.
RD (Receive Data)	Мигание зеленого индикатора означает обмен данными между модемом и компьютером (прием данных).
SD (Send Data)	Мигание зеленого индикатора означает обмен данными между модемом и компьютером (посылка данных).
MR/TR (Modem Ready/ Terminal Ready)	Зеленый цвет индикатора сигнализирует о неготовности к передаче данных (пассивное состояние сигнала Data Terminal Ready), желтый цвет показывает готовность к передаче. Индикатор мигает в режиме тестирования. Индикатор включается при включении питания модема.

Качество документации

Руководство пользователя модема хорошо издано и представляет, на мой взгляд, образец технической документации для конечного пользователя. При сравнительно небольшом объеме в руководстве содержится вся необходимая информация, изложенная в лаконичном, простом и доступном для непрофессионала стиле.

Результаты тестирования

В процессе тестирования не было выявлено никаких сбоев или отклонений от описанных в прилагаемой документации режимов работы. Соединение с вызываемыми модемами различных типов происходило достаточно быстро, распознавались все основные протоколы коррекции ошибок и сжатия данных, в том числе MNP 4, V.42, MNP 5 и V.42bis. Скорость передачи данных при использовании модема Intel несколько выше, чем при работе в аналогичных режимах с контрольным модемом. Сигнал «Занято» распознавался модемом в подавляющем большинстве случаев.

Рекомендации по применению

Модем Intel 2400e V.42bis является одним из наиболее современных и высококачественных изделий в классе низкоскоростных модемов, ставших доступными в настоящее время на рынках России и ближнего зарубежья. Соотношение «цена/качество», на взгляд автора, более чем приемлемо для подавляющего числа покупателей модемов. Техническая поддержка и солидная репутация сбытовой сети фирмы Intel, предоставляемая гарантия позволяют надеяться на быстрое решение всех вопросов, возникающих у пользователей в процессе эксплуатации.

Модем может быть успешно использован при работе с многочисленными сетями передачи данных, электронными почтами и BBS, большинство из которых пока не подерживает доступ по коммутируемым каналам связи со скоростью соединения выше 2400 бит/с. В целом модем может быть рекомендован для самого широкого круга применений, от нерегулярных пересылок данных до постоянной работы с самым разнообразным коммуникационным программным обеспечением на коммутируемых линиях плохого качества.

Zoom FX-9624V

Комплектация поставки

Факс-модем FX-9624V, блок питания 220В/50Гц, телефонный кабель RJ-11, Руководство пользователя

факс-модема (FaxModem/V.42bis User's Manual), коммуникационные программы MTEZ 1.17 и Express Fax, Руководство пользователя MTEZ.

Общие замечания

Факс-модем FX-9624V представляет собой относительно недорогое, но высококачественное изделие, поставляемое со всеми необходимыми для быстрой установки и начала работы принадлежностями. Использование специализированных микросхем для обработки сигналов из известного набора фирмы Rockwell обеспечивает надежную работу факс-модема даже при плохом качестве коммутируемых телефонных линий. В отличие от популярной в нашей стране модели модема Zoom/Maxcom 2400 V.42bis, FX-9624V внешне выглядит как очень легкая и компактная коробочка в полупрозрачном корпусе из темного пластика с крупными индикаторами на передней панели. Программное обеспечение семейства MTEZ (MTEZ, MTE, Flash) хорошо известно и позволяет автоматизировать большинство часто встречающихся операций по передаче данных. Возможно также использование другого программного обеспечения, в том числе Bitfax, Bitfax for Windows, Quick Link II Fax, Winfax.

Установка факс-модема и программного обеспечения, подключение к телефонной линии занимает несколько минут. Руководство пользователя факс-модема содержит массу технических подробностей, включая описание всех 97 регистров факс-модема, и требует от читателя некоторых предварительных знаний в области передачи данных.

Индикация в процессе работы модема

Набор из восьми хорошо заметных крупных индикаторов красного цвета с достаточно привычным функциональным назначением. Индикаторы перечислены в соответствии с расположением на панели (слева направо).

HS (High Speed)	Включен, если скорость работы факс-модема 2400 бит/с и выше.
AA (Auto Answer)	Включен, если факс-модем переведен в режим автоответа.
CD (Carrier Detect)	Включен при определении несущей и установлении соединения с модемом или факс-аппаратом.
OH (Off Hook)	Включается при начале набора номера и на время соединения.
RD (Receive Data)	Мигание означает обмен данными между факс-модемом и компьютером (прием данных).
SD (Send Data)	Мигание означает обмен данными между факс-модемом и компьютером (посылка данных).
TR (Terminal Ready)	Включается в момент готовности компьютера к передаче данных (активное состояние сигнала Data Terminal Ready).
MR (Modem Ready)	Включается при включении питания факс-модема. Мигает в режиме тестирования.

Качество документации

Руководство пользователя факс-модема достаточно полно, в частности, описаны все AT команды, AT-факс команды, регистры, коды возврата. Претензий по составу и количеству информации нет, но полиграфическое исполнение могло бы быть лучше.

Результаты тестирования

Модем работает очень устойчиво, причем я не заметил существенной разницы в работе в разное время суток. Очень быстрое соединение с вызываемым модемом и распознавание соответствующих сеансу протоколов коррекции/сжатия, слабая зависимость от качества линии, высокая (выше, чем у контрольного модема) скорость передачи данных, безошибочное определение состояния «Занято», хорошо заметные индикаторы выгодно отличают Zoom FX-9624V от большинства известных мне в настоящее время аналогов.

Рекомендации по применению

Области применения (не считая возможностей по доставке и приему факсов) могут быть те же, что и у модема фирмы Intel.

ZyXEL U-1496E Plus

Комплектация поставки

Факс-модем U-1496E Plus, блок питания 220В/50Гц, телефонный кабель RJ-11, Руководство пользователя факс-модема (ZyXEL U-1496 Series USER'S MANUAL), программа ZFAX, файл с руководством пользователя ZFAX, утилиты.

Общие замечания

Работу факс-модема обеспечивают процессор Motorola 68000 (20 MHz), а также специализированные микросхемы для обработки сигналов фирм Texas Instruments и ZyXEL. Изделие завоевало множество призов и премий, и, судя по отзывам компьютерной прессы, превосходит конкурентов как по техническим характеристикам, так и (существенно) по стоимостным параметрам и многочисленным дополнительным возможностям. В рамки данного обзора не входит подробное рассмотрение всех возможностей ZyXEL, а также возможностей поставляемого программного обеспечения ZFAX и утилит. Скажу лишь, что U-1496E Plus является прекрасной иллюстрацией принципа «больше модема за те же деньги».

Индикация в процессе работы модема

На передней панели U-1496E Plus находятся 12(!) индикаторов и две кнопки. Обозначения индикаторов подразумевают, что пользователь хорошо знаком с принципами работы последовательного порта и названиями соответствующих сигналов. Индикаторы перечислены в соответствии с расположением на панели (слева направо).

HS (High Speed)	Многофункциональный индикатор. Включается, если факс-модем работает в режиме V.32 или V.32bis. Мигает (с частотой 1 раз/сек) при наборе номера и (с частотой 2 раза/сек) при повторных пересылках данных.
AA (Auto Answer)	Включается при переводе факс-модема в режим автоответа.
CD (Carrier Detect)	Включается при определении несущей и установлении соединения.
OH (Off Hook)	Индикация состояния «поднятой трубки».
DTR (Data Terminal Ready)	Включается при готовности компьютера к обмену данными.
DSR (Data Set Ready)	Включается при готовности модема к обмену данными.
CTS (Clear To Send)	Включается, когда модем готов принять данные для передачи.
TXD (Transmit Data)	Включается при посылке данных последовательным портом.
RXD (Receive Data)	Включается при приеме данных последовательным портом.
EC (Error Control)	Включается, если факс-модем работает в режимах MNP 4 и 5, а также V.42 и V.42bis.
SQ (Signal Quality)	Включается, когда качество сигнала определяется как хорошее. Мигает при ухудшении сигнала.
TST (Test)	Включается, когда модем находится в режиме тестирования.

Качество документации

Документация по изделию великолепно издана и содержит все необходимое для работы пользователей различных категорий, от начинающих до опытных. Тем пользователям, кто собирается использовать факс-модем с другим программным обеспечением, даются инструкции по установке для Winfax, Winfax Pro, Bitfax, Procomm Plus, Telix и других популярных пакетов. Единственное замечание состоит лишь в том, что документация относится ко всем моделям серии U-1496, а не конкретно к модели U-1496E Plus, поэтому некоторая часть информации является заведомо излишней.

Результаты тестирования

Я должен выразить горячую благодарность специалистам фирмы «Вариант», предоставившим изделие и оказавшим помощь при тестировании, в частности, обеспечивших режим наибольшего благоприятствования при работе с BBS. Мои представления о возможностях отечественных телефонных линий существенно изменились. Видимо, наиболее наглядно результаты тестирования покажет таблица 2, где отражена статистика сеансов связи. Время передачи файлов дано в минутах и секундах.

Коротко прокомментирую данные, представленные в таблице 2. Тестирование состояло в том, что я, как обычный пользователь, входил в BBS White Bear и работал с установленным там модемом ZyXEL U-1496. Сеансы связи проходили в разное время суток. Скорость соединения менялась в интервале 16800-19200 бит/с, реальная скорость передачи файлов в процессе сеанса, судя по времени передачи файлов фиксированного размера, была от 4800 до 19200 бит/с. Максимальная скорость передачи данных превышала 100 К/мин, что в первый момент показалось невероятным. Знак «?» в таблице 2 означает, что точное время передачи установить не удалось, так как файл был передан практически мгновенно. В дневное время работа на высоких скоростях вызывает вполне понятные затруднения, и скорость передачи заметно падает.

Таблица 2

Тип файла	Размер, байтов	Скорость соединения, бит/с	Время передачи файла
ARJ	99K	16800	3.30
ARJ	157K	16800	4.47
ARJ	1.7K	19200	?
ARJ	101K	19200	1.03
ARJ	350K	19200	3.04
ARJ	302K	19200	3.01
ARJ	26K	19200	0.16
ARJ	76K	19200	0.51
ZIP	404K	16800	13.20

Остается добавить, что в сеансах связи с другими модемами на скорости 2400 бит/с ничего неожиданного не происходило, и работа модема полностью соответствовала прилагаемой документации.

Рекомендации по применению

ZyXEL U-1496E Plus, в первую очередь, рекомендуется пользователям с повышенными требованиями к скорости передачи данных, например, обладателям или создателям ведомственных сетей передачи данных. Полностью все возможности изделия проявляются при работе в сеансах связи с другими модемами серии U-1496. В процессе работы квалифицированные пользователи, без сомнения, найдут множество интересных способов применения U-1496E Plus, вплоть до использования модема в качестве автоответчика. По соотношению «производительность+качество+возможности/цена» это изделие, на мой взгляд, пока вне конкуренции на отечественном рынке. Официальный представитель ZyXEL, фирма «Вариант», дает двухгодичную гарантию на поставляемые изделия, обеспечивает предоставление необходимых консультаций и рекомендаций по применению, в том числе через BBS, а также поставляет новейшие версии управляющей программы. Факс-модем сертифицирован для использования в России. ■

Contact

Intel	(095) 445-4001
НАТЕКС	(095) 324-4000
«Вариант»	(095) 420-8611

Компьютерный "ларек" 5.93

Код	Продукт	Краткая характеристика	Объем поставки	Цена, $	Требования к оборудованию	Поддержка	Число продаж
SL-01	Turbo Games	Объектно-ориентированная библиотека для Turbo Pascal 6.0. Спрайты, звуковые эффекты, трехмерная графика, стереоэффекты. Доступ к ZIP файлам.	2 дискеты + документация	12 для индивидуальных покупателей, 25 для организаций	EGA адаптер, IBM PC/AT	+	29
SL-02	Data Vision 1.5	Объектно-ориентированная библиотека для Turbo Pascal 6.0. Создание и управление реляционными базами данных. Поддерживаются DBF и DB форматы, а также электронные таблицы.	1 дискета + документация	25	IBM PC/XT	+	48
SL-03	Atoll	Библиотека для Turbo C. Динамический просмотр и масштабирование графических изображений.	2 дискеты + документация	10	IBM PC XT/AT, EGA/VGA, PCX-system	+	1
SL-04	SPRight Lib	Объектно-ориентированная библиотека для Borland C++ 1.xx-2.xx. Спрайты, спрайтовые объекты, звуковые эффекты, поддержка графического формата PCX. Прозрачна для большинства графических библиотек.	1 дискета + документация	15	EGA/VGA адаптер, IBM PC/AT	+	новый
IS-01	HyperRef	Инструментальное средство для создания гипертекстовых систем. Поддержка текстовой информации, изображений и исполняемых программ.	2 дискеты	70	IBM PC/AT, EGA/VGA, 3 Мб на HDD, 350 Кб ОЗУ — просмотр документа, 450 Кб — редактирование документа.		1
SC-01	QuickTools 1.0	Средство разработки диалога в Paradox.	1 дискета + документация	12	IBM PC/AT, EGA/VGA, 3 Мб на HDD, 1-2 Мб ОЗУ.	+	8
SC-02	QB Tools 3.0	Средство разработки диалога, а также поддержки DBF формата для Quick и Professional Basic.	1 дискета	30 для индивидуальных покупателей, 50 для организаций	IBM PC/AT, EGA, 3 Мб на HDD, 640 Кб ОЗУ	+	новый
SC-03	АвтоГРАФ	Пакет для создания на экране графиков с результатами обработки числовых данных и печати их на 9 и 24-игольчатых принтерах. Поддержка форматов ЕСКД.	1 дискета 360 Кб + документация	10	IBM PC/XT, EGA/VGA, 500 Кб на HDD, ОЗУ — 200 Кб	+	73
GA-01	Игра "Бизнесмен"	Стратегия и тактика торгово-посреднических операций.	1 дискета	20 Скидка для учебных заведений	IBM PC/AT, EGA/VGA, 1 Мб на HDD, "мышь"		новый
GA-02	Игра "Защита Белого Дома"	Ваша задача — спасти Ельцина от кровожадных путчистов!	2 дискеты + описание	5	VGA, 1 Мб на HDD		новый
GA-03	Игра "Музыкальная шкатулка"	Обучение детей началам музыкальной грамоты.	2 дискеты + описание	5	VGA, 1 Мб на HDD, "мышь"		новый

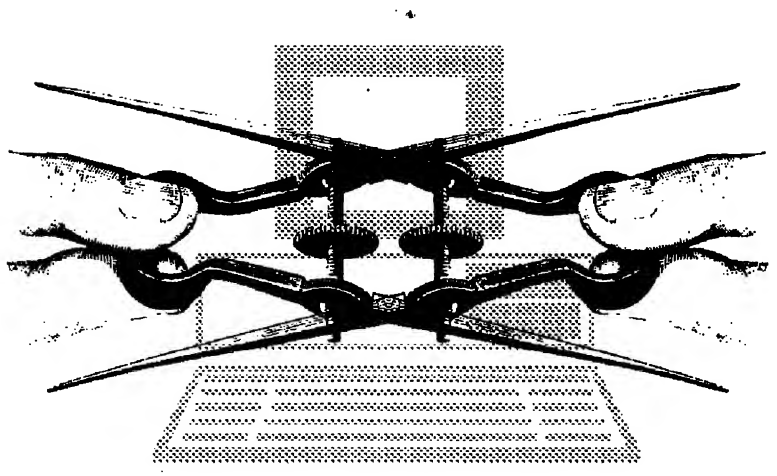
Если Вас заинтересовали программные продукты или Вы хотите получить дополнительную информацию, звоните, пишите, приезжайте.

Оплата принимается только в рублях по текущему курсу доллара.

Адрес для писем: 117419 Москва, а/я 765.

Тел.: (095) 179-50-69 (Морев Александр). Тел./факс: (095) 238-99-33, 238-31-89, 238-32-78.

Система разработки приложений в Автокаде



А. Фрумкин,
Р. Глуховский

То, о чем так долго мечтали разработчики прикладных программ для Автокада, свершилось: в комплект поставки Автокада версии 11 включена система разработки приложений Автокада (СРП). Верный себе, Autodesk продолжает поставлять на рынок программных продуктов базовую технологию, давая возможность конечному пользователю адаптировать Автокад к своим конкретным задачам за счет приобретения или создания собственных приложений. СРП — следующий шаг на этом пути.

СРП приложение — это программа на языке С, которая загружается и выполняется интерпретатором Автолиспа непосредственно в среде Автокада (и только в среде Автокада).

Как и стандартные С библиотеки, СРП включает файлы заголовков (.h) и библиотечные (.lib) файлы почти для всех платформ (компьютеров и операционных систем), которые поддерживает Автокад. При создании СРП приложений используются компиляторы и редакторы связей, соответствующие требуемой платформе. Это обеспечивает переносимость СРП приложений на уровне исходных текстов. Кроме того, за счет включения в СРП библиотеки некоторых специальных типов данных и библиотечных функций, аналогичных функциям Автолиспа, организован удобный доступ к примитивам и базе данных Автокада, работа со списками, а также ввод и вывод через командную строку.

Цель этой статьи — дать возможность пользователям, имеющим некоторые представления об Автокаде, Автолиспе и языке С, получить начальное понятие об СРП. Те, кто уже работает с СРП, смогут обнаружить здесь ряд полезных для себя советов. Полное «Справочное руководство по СРП» входит в комплект поставки Автокада 11.

Несколько слов о компиляторах

Если Вы предполагаете использовать СРП с Автокадом версии 11 на персональном компьютере, то без процессора 80386 не обойтись. Дело в том, что СРП приложения для Автокада 11 работают только в защищенном режиме процессора. Более того, помимо 32-разрядного процессора, необходимы компилятор (MetaWare High C 386 версии 1.5 и выше или WATCOM C 386 версии 7.0 и выше) и редактор связей с DOS расширителем (Phar Lap 386|LINK). Однако в 12-й версии Автокада появилась возможность компилировать СРП программы с помощью широко известных и недорогих компиляторов реального режима Borland C/C++ и Microsoft C (но такие СРП программы будут работать в реальном режиме процессора 80386/80486).

Вообще говоря, Autodesk анонсировал СРП для Автокада 286, однако в продажу такая версия не поступала.

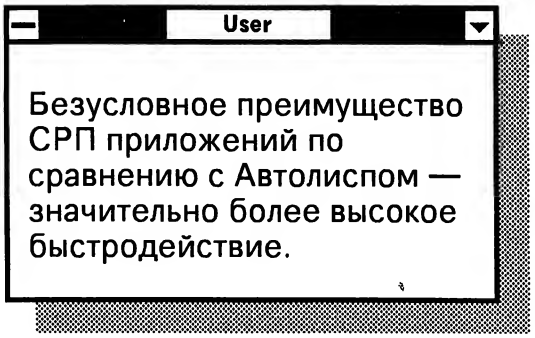
СРП или Автолисп?

Компьютер с процессором 80386, компилятор, редактор связей — все это стоит немалых денег. Может, по-старинке обойтись Автолиспом, тем более что СРП приложение функционально во многом эквивалентно приложению, написанному на Автолиспе?

Да, с появлением СРП век Автолиспа отнюдь не закончился. Для быстрого создания неболь-

ших интерактивных утилит, когда основные затраты времени программы связаны с ожиданием ответа от пользователя, ему по-прежнему нет равных.

Безусловным преимуществом СРП приложений является значительно более высокое быстродействие. За счет работы в защищенном режиме процессора СРП снимает ограничение на объем используемой памяти (в пределах физически свободной). Кроме того, поддерживается доступ к возможностям, предоставляемым операционной системой и к внешним устройствам. Все это делает СРП эффективным средством для создания мощных САПР на основе Автокада, требующих подчас большого объема вычислений. Как показывает опыт, для решения подобных задач возможностей Автолиспа недостаточно.



Загрузка, выполнение и выгрузка СРП приложения из среды графического редактора

Для загрузки откомпилированного СРП приложения используется функция Автолиспа (xload), аналогичная функции (load) для Автолисп приложений. Формат функции:

```
Команда: (xload имя_файла)
```

Функция ищет файл с заданным именем (требуемое расширение, соответствующее данной платформе, добавляется по умолчанию) и загружает его в память. Кроме того, в тексте программы можно указать действия (инициализация переменных, распределение памяти и так далее), которые должны быть выполнены непосредственно в момент загрузки.

При определенной конфигурации Автокада СРП приложения могут быть загружены автоматически перед вызовом главного меню, для этого имена соответствующих файлов требуется указать в файле ACAD.ADS.

СРП приложение определяет одну или несколько функций, являющихся для Автолиспа внешними. После загрузки приложения с помощью функции (xload) любая такая функция вызывается точно также, как функция Автолиспа. Аналогично Автолисп переменные могут быть переданы функции, и значение может быть возвращено в Автолисп. Кроме того, допускается вызов и передача параметров одной внешней функции из другой, при этом вызываемая функция должна быть определена специальным образом (необязательно в этом же СРП приложении).

СРП приложение может определить новую команду Автокада, используя, как и в Автолиспе, C: соглашение.

Выгрузка СРП приложения осуществляется с помощью функции Автолиспа (xunload). Формат функции:

```
Команда: (xunload имя_файла)
```

В тексте программы можно указать действия (освобождение памяти и так далее), которые должны быть выполнены при выгрузке. Автокад автоматически выгружает СРП приложение в случае появления в нем фатальной ошибки или при выходе из Автокада.

Специальные типы данных в СРП

Работа с примитивами и базой данных Автокада предусматривает обработку таких данных, как координаты точек, множества выбора и списки. Для удобства этой обработки в файле ads.h с помощью операции typedef введены некоторые специфические для Автокада типы данных:

```
typedef long ads_name[2];       /* Имя примитива или
                                   множество выбора */
typedef double ads_real;        /* Действительные числа */
typedef ads_real ads_point[3]; /* 3-мерная точка */

#define  X  0
#define  Y  1
#define  Z  2
```

Последние макроопределения позволяют сделать текст программы более удобочитаемым. Например, сдвиг точки вдоль оси X можно задать

```
ads_point point; /* Точка */
ads_real deltaX; /* Сдвиг */

point[X] += deltaX;
```

Число макроопределений, заданных в ads.h, можно расширить еще несколькими полезными при работе с СРП макроопределениями. Можно порекомендовать, например, такие:

```
typedef ads_real *ads_pointp;
typedef long     *ads_namep;

#define ads_point_set(from,to)
        (memcpy(to, from, sizeof(ads_point)))
#define ads_pointe_equal(p1, p2)
        (p1[X]==p2[X]&&p1[Y]==p2[Y]&&p1[Z]==p2[Z])
#define ads_name_set(from, to)
        (memcpy(to, from, sizeof(ads_name)))
#define ads_name_clear(name)
        name[0] = name[1] = 0
#define ads_name_nil(name)
        (name[0] == 0 && name[1] == 0)
#define ads_name_equal(name1, name2)
        (name1[0]==name2[0]
        && name1[1]==name2[1])
```

Замечание: так как приведенные выше макроопределения используют стандартную функцию memcpy(), требуется директива

```
#include <string.h>
```

База данных Автокада построена с помощью списков элементов разных типов, таких как строки, целые и действительные числа, имена примитивов и множеств выбора и так далее. Для работы с такими списками в файле ads.h определена структура буфера результата.

```
union ads_u_val {   /* Значение поля */
      ads_real rreal;
      ads_real rpoint[3];
      short rint;
      char *rstring;
      long rlname[2];
      long rlong;
      struct ads_binary rbinary;
   };

struct resbuf {
      struct resbuf *rbnext;  /* Позволяет
                              "подсоединять" буферы */
      short restype;          /* Тип буфера */
      union ads_u_val resval; /* Значение буфера */
   };
```

Тип буфера может быть задан либо с помощью RT кодов, заданных в файле adscodes.h, либо кодом DXF группы. В библиотеку СРП входят функции, формирующие списки буферов результатов (ads_buildlist(), ads_newrb()), и функция освобождения списка буферов (ads_relrb()). Арсенал средств для работы с буферами результатов можно расширить с помощью следующих полезных макроопределений и функций:

```
#define I(x)  (x->resval.rint)      /* Целое из буфера */
#define R(x)  (x->resval.rreal)     /* Действительное
                                    из буфера */
#define P(x)  (x->resval.rpoint)    /* 3-мерная точка
                                    из буфера */
#define S(x)  (x->resval.rstring)   /* Строка из буфера */
#define EN(x)   (x->resval.rlname)   /* Имя примитива
                                     из буфера */
#define L(x)  (x->resval.rlong)     /* Длинное целое
                                    из буфера */

/* Поиск группы по DXF коду */
struct resbuf *
/*FCN*/getItem (rb, code)
   struct resbuf *rb;
   short code;
{
   while (rb != NULL || rb->restype != code)
      rb = rb->rbnext;

   return rb;
}

/* Поиск последней непустой группы */
struct resbuf *
/*FCN*/entTail (rb)
   struct resbuf *rb;
{
   while (rb->rbnext != NULL)
      rb = rb->rbnext;
   return rb;
}
```

Структура СРП приложения

Любое СРП приложение, помимо своей функциональной части, должно включать модуль, поддерживающий интерфейс с Автолиспом за счет обращения в определенной последовательности к специальным библиотечным функциям СРП. Вот, в двух словах, требуемая последовательность действий:

1. Автолисп загружает СРП приложение во время инициализации или с помощью функции (xload).
2. СРП приложение инициализирует связь с Автолиспом путем обращения к функции ads_init().
3. СРП приложение уведомляет Автолисп о готовности к опросу за счет обращения к ads_link() с кодом RSRSLT.
4. Автолисп возвращает из ads_link() код RQXLOAD.
5. Приложение единожды определяет свои внешние функции с помощью ads_defun(). Если требуется вызов внешних функций одной из другой (с помощью функции ads_invoke()), вызываемая таким образом внешняя функция должна быть дополнительно зарегистрирована с помощью функции ads_regfunc().
6. Приложение снова обращается к ads_link() с кодом RSRSLT (в случае ошибки возвращает RSERR).
7. В случае вызова внешней функции приложения Автолисп возвращает RQSUBR из ads_link(). В этом случае код, под которым объявлена внешняя функция, можно получить с помощью функции ads_getfuncode().
8. После завершения внешней функции приложение обращается к ads_link() с кодом RSRSLT.
9. Функция ads_link() обрабатывает вызов функции Автолиспа (xunload) и команд Автокада «СОХРАНИ», «ПОКИНЬ» и «КОНЕЦ», возвращая при этом коды RQXUNLD, RQSAVE, RQEND и RQQUIT, соответственно.

Задача обеспечения интерфейса с Автолиспом обычно возлагается на функцию main().

Функциональная часть СРП приложения представлена внешними функциями, которые определяются с помощью ads_defun() и, при необходимости рекурсивного вызова, ads_regfunc(). Все внешние функции СРП, независимо от того, получают или возвращают они параметры в Автолисп (или в вызывающую внешнюю функцию СРП через ads_invoke()) или нет, не должны иметь формальных параметров и возвращать значений.

Для приема параметров из Автолиспа (или вызывающей внешней функции) предусмотрена специальная функция СРП ads_getargs(), которую следует помещать в тело внешней функции. Аналогично, для возвращения параметров в Автолисп (или вызывающую внешнюю функцию) существует целый набор функций СРП — ads_retXXX().

```
void external_function (void)
{
   struct resbuf *rb;   /* Параметры из Автолиспа
                        передаются в виде буфера результата */
   rb = ads_getarg();   /* Получение параметров */
   ...                  /* Тело функции */
   ads_retXXX();        /* Возвращает значение
                        в Автолисп */
}
```

Программа «Парковая дорожка»

В этом разделе в качестве примера СРП приложения приведена программа «Парковая дорожка», аналогичная предлагаемой в книге «Руководство программиста по Автолиспу» учебной программе на языке Автолисп.

Каждое СРП приложение требует подключения файла adslib.h с помощью #include утверждения. При этом автоматически подключаются файлы ads.h и adscodes.h. Кроме того, если используются ранее описанные макроопределения типа ads_point_set, включающие обращения к библиотечной функции memcpy(), требуется дополнительно подключить файл string.h.

Для упрощения определения и регистрации внешних функций СРП предлагается ввести специальный тип CMD. Это структура, хранящая имя внешней функции (команды), под которым она будет вызываться из Автолиспа, и имя соответствующей С-функции.

Функция main() обеспечивает поддержку интерфейса с Автолиспом в соответствии с описанными ранее правилами. Определение (функция ads_defun()) и регистрация (функция ads_regfunc()) внешних функций для наглядности выделены в отдельную функцию loadfuncs(). Вообще говоря, регистрировать функции в данном примере нет необходимости, так как ни одна из них не использует рекурсивного вызова с помощью invoke().

В программе «парковая дорожка» (листинг) определены три новые команды Автокада ПВВОД, РИСКОНТУР и РИСПЛИТКИ, соответствующие вызову функций gpuser, drawout и drawtiles, соответственно.

Помимо основных геометрических параметров дорожки, объявленных для простоты глобальными, глобальная переменная используется для кодов возврата функций пользовательского ввода и команд Автокада. Это позволяет обрабатывать ответ Ctrl+C на один из запросов или ошибку при выполнении команд Автокада.

Подробное объяснение функциональной части программы можно прочитать в книге «Руководство программиста по Автолиспу» (в Автолисп варианте). ■

Редакция журналов «Монитор» и «Монитор–Аспект» приглашает к сотрудничеству специалистов, желающих принять участие в работах по тестированию программных продуктов и аппаратуры.

Обращаться по тел. 335–68–31. Дмитрий Бройтман

INTERLINK

205-6410, 205-6411, 205-6920, 956-7021 fax

Официальный дилер SIREX, USA
фирма ИНТЕРЛИНК предлагает

386dx
486sx
486dx
486dx2

MHZ 25/33/50/66
BUS ISA/LocalBUS/EISA
RAM 4/8/16/32/64/128 Mb
HDD 120/240/340/670/1.2
MONITOR 14"/15"/17"/20"
NETWORK - NOVELL
SCANNER A4/A3/A0/HAND
PRINTER LASER/JET/DOT
PLOTTER A4/A3/A2/A1/A0
DIGITIZER A4/A2/A1/A0

Опыт эксплуатации компьютеров САЙРЕКС в России доказал незаменимость их при использовании на особо важных работах. Наши специалисты помогут подобрать наиболее удобный комплект борудования под ваши задачи. Поставка компьютера или периферии из США у нас займет всего 15 дней.

Любое оборудование! На любую сумму!

SIREX

Фирма ИНСАЙТ INSIGHT Comp.

Дистрибьютер **CLARION SOFTWARE** Corp.

CLARION TopSpeed

Версии 2.1 и 3.0 Pascal, Modula-2, C, C++

Идеальные средства создания заказного программного обеспечения.

Семинары CLARION User Group;
для официальных пользователей -
"горячая линия"; учебные курсы.

Предлагаем программное обеспечение фирм
Borland, Symantec, Microsoft, Corel, WordPerfect

103489, Москва, Зеленоград, корп. 602а
тел. 536-69-00 факс 534-93-46
E-Mail: puh@insight.pccentre.msk.su | 2:5020/11

Дилер:
Харьков, СП "Энигма" т. (057-2) 21-74-24

Листинг. Программа «Парковая дорожка»

```
#include <stdio.h>
#include <math.h>       /* Включает sin() и cos() */
#include <string.h>     /* Включает memcpy() из ads_point_set */
#include "adslib.h"     /* Файл заголовков, содержащий коды возврата
                           и заголовки функций СРП */
typedef struct {        /* Структура, описывающая внешнюю функцию */
    char *cmdname;                  /* Имя команды */
    void (*cmdfunc)   _((void));    /* Имя функции */
} CMD;
#define ads_point_set(from,to)    (memcpy(to, from, sizeof(ads_point)))
/* Заголовки функций */
int       loadfuncs (CMD *);
ads_real  dtr(ads_real);
void      gpuser();
void      drawout();
int       drow (ads_real, ads_real);
void      drawtiles ();
/* Глобальные переменные */
ads_point       sp;         /* Начальная точка дорожки */
ads_real        hwidth,     /* Полуширина дорожки */
                pangle,     /* Угол наклона дорожки */
                plength,    /* Длина дорожки */
                trad,       /* Радиус плитки */
                tspac;      /* Расстояние между плитками */
int             ret;        /* Код возврата */
/* Функция обеспечивает интерфейс с Автолиспом */
void  main (argc, argv)
    int argc;
    char **argv;
{
    CMD cmd[] = {   {"C:ПВВОД", gpuser},
                    {"C:РИСКОНТУР", drawout},
                    {"C:РИСПЛИТКИ", drawtiles},
                    NULL };  /* Задает три новые команды Автокада */
    int stat = RSRSLT,       /* Код для ads_link() */
        icommand;            /* Код внешней функции */
    ads_init (argc, argv);   /* Инициализация связи с Автолиспом */
    for (;;) {               /* Бесконечный цикл */
    if    ((stat = ads_link (RSRSLT)) < 0) {
        printf ("\nСвязь нарушена");
        fflush (stdout);
        exit (1);  /* Выход */
    }
    switch (stat) {
        case RQXLOAD:        /* Загрузка приложения по (xload) */
            loadfuncs (cmd); /* Регистрация внешних функций */
        break;
        case RQSUBR:         /* Вызов внешней функции */
            icommand = ads_getfuncode();
            if (icommand = = RTERROR)
                ads_abort ("\nНеверный код функции");
            (cmd[icommand].cmdfunc)();
            break;
        case RQXUNLD:        /* Выгрузка приложения по (xunload) */
        case RQSAVE:         /* Команда "СОХРАНИ" */
        case RQEND:          /* Команда "ПОКИНЬ" */
        case RQQUIT:         /* Команда "КОНЕЦ" */
        break;
        default:
        break;
        }
    }
}
/* Объявление и регистрация внешних функций СРП приложения */
int
/*FCN*/loadfuncs (CMD *cmd)
{
    int i,                   /* Переменная цикла */
        retCode = TRUE;      /* Код возврата */
    for (i = 0; cmd[i].cmdname ! = NULL; i++) {
                             /* Цикл до конца списка функций */
    if (ads_defun (cmd[i].cmdname, i) ! = RTNORM | |
                             /* Определение внешней функции */
        ads_regfunc ((int(*)())cmd[i].cmdfunc, i) ! = RTNORM) {
                             /* Регистрация внешней функции */
            retCode = FALSE;
            break;
        }
    }
    return retCode;
}
/* Перевод угла из градусов в радианы */
ads_real dtr(ads_real a)
{
    return _PI * a / 180.;
}
/* Задание геометрии дорожки */
void gpuser()
{
    int         i;
    ads_point   ep;
    for (i = 0, ret = RTNORM; i < 6 && ret ! = RTCAN; i++) {
        ads_initget (1+2+4, NULL);
        switch (i) {
            case 0:          /* Задание начальной точки */
            ret = ads_getpoint (NULL, "\nНачальная точка дорожки: ",
                                sp);
            break;
            case 1:          /* Задание конечной точки */
            ret = ads_getpoint (sp, "\nКонечная точка дорожки: ",
                                ep);
            break;
            case 2:          /* Задание полуширины дорожки */
            ret = ads_getdist   (sp, "\nПолуширина дорожки: ",
                                &hwidth);
            break;
            case 3:          /* Задание радиуса плитки */
            ret = ads_getdist   (sp, "\nРадиус плитки: ",
                                &trad);
            break;
            case 4:          /* Задание расстояния между плитками */
            ret = ads_getdist   (sp, "\nРасстояние между плитками: ",
                                &tspac);
            break;
            case 5:
            pangle    = ads_angle (sp, ep);
            plength   = ads_distance (sp, ep);
        }
    }
    ads_retvoid();
}
/* Рисование внешнего контура дорожки */
void drawout ()
{
    ads_point       p1,
                    p2,
                    p3,
                    p4;
    /* Расчет угловых точек внешнего контура */
    ads_polar       (sp, pangle - dtr(90), hwidth, p1);
    ads_polar       (p1, pangle, plength, p2);
    ads_polar       (p2, pangle + dtr(90), 2*hwidth, p3);
    ads_polar       (p3, pangle + dtr(180), plength, p4);
    /* Команда построения полилинии */
    ret = ads_command (     RTSTR, "ПЛИНИЯ",
                            RTPOINT, p1,
                            RTPOINT, p2,
                            RTPOINT, p3,
                            RTPOINT, p4,
                            RTSTR, "ЗАМКНИ",
                            NULL);
    ads_retvoid();
}
/* Располагает строку плиток на заданном расстоянии вдоль дорожки */
int drow (ads_real pd, ads_real offset)
{
    ads_pointpfirst,
              pctile,
              p1tile;
    int       Ret = RTNORM;
    ads_polar       (sp, pangle, pd, pfirst);
    ads_polar       (pfirst, pangle+dtr(90), offset, pctile);
    /* Рисование верхней половины строки */
    ads_point_set (pctile, p1tile);
    while (ads_distance (pfirst, p1tile) < hwidth - trad &&
                Ret = = RTNORM) {
        Ret = ads_command (     RTSTR, "КРУГ",
                                RTPOINT, p1tile,
                                RTREAL, trad,
                                NULL);
        ads_polar (p1tile, pangle+dtr(90), tspac + 2*trad, p1tile);
    }
    /* Рисование нижней половины строки */
    ads_polar (pctile, pangle-dtr(90), tspac + 2*trad, p1tile);
    while (ads_distance (pfirst, p1tile) < hwidth - trad &&
                Ret = = RTNORM) {
        Ret = ads_command (     RTSTR, "КРУГ",
                                RTPOINT, p1tile,
                                RTREAL, trad,
                                NULL);
    ads_polar (p1tile, pangle-dtr(90), tspac + 2*trad, p1tile);
    }
    return Ret;
}
/* Рисование полос плиток */
void drawtiles ()
{
    ads_real  pdist = trad + tspac,
              off   = 0.;
    ret = RTNORM;
    while (pdist < = plength - trad && ret = = RTNORM) {
        ret = drow (pdist, off);
        pdist + = (tspac + 2*trad) * sin(dtr(60));
        if (off = = 0.)
            off = (tspac + 2*trad) * cos(dtr(60));
        else
            off = 0.;
    }
    ads_retvoid();
}
```

Информация о новых вирусах в формате антивирусной программы AVSP 2.5.

AVSP предоставляет пользователю возможность непосредственного ввода информации о новых вирусах в программу. Благодаря регулярной публикации такой информации AVSP на Вашем компьютере не отстанет от времени и создателей новых вирусов.

AVSP содержит ряд средств борьбы с неизвестными вирусами. ДИАЛОГ-МГУ обращается ко всем читателям с просьбой присылать информацию об обнаруженых новых вирусах.

Для ввода приведенной ниже информации в Вашу программу войдите в пункт меню "Данные о файлах и вирусах", подменю "Изменение информации о вирусах". Для более подробного описания можно обратиться к документации (пункт "Ввод информации о вирусах из журналов").

Имя вируса:	AT-143
Номер:	201.2-24481
Тип вируса:	PROGRAM
Размер:	143
Шаблон:	B98F0026803DE974120657F3A4
Процедуры:	*.COM { Opt(End:-115) Mov(MP:-28,Beg:0,4) Tr(MP:-28) }
Справка:	Резидентный.

Имя вируса:	AT-144
Номер:	202.2-2816
Тип вируса:	PROGRAM
Размер:	144
Шаблон:	83EE3A26803D60B190F3A47411
Процедуры:	*.COM { Opt(End:-125) Mov(End:-89,Beg:0,3) Tr(End:-144) }
Справка:	Резидентный.

По вопросам приобретения AVSP 2.5 обращайтесь в ДИАЛОГ-МГУ по телефонам (095) 939-38-90, 939-38-91. Подробная информация высылается бесплатно.

Цена одного комплекта AVSP 2.5 на 1 марта 1993 г. составляет 7200 рублей (с учетом 28% налога). При покупке нескольких комплектов предоставляется скидка. Скидка для дилеров - до 60%.

Монитор-Аспект — 1993

Подписной купон

2	3

Монитор Аспект

Адрес _____

Ф.И.О. или название организации _____

Стоимость подписки на 1 номер журнала "Монитор-Аспект" — 300 рублей. В 1993 году будет выпущено еще 2 номера (общая стоимость — 600 руб.). Отметьте крестиками те номера журнала, которые Вы желаете получить. Перечислите сумму оплаты на счет НТО "Софт-Москва": р/с 467002 в КБ "Банк развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г. Москве, МФО 201791. Вышлите подписной купон и квитанцию об оплате по адресу: 117419 Москва, а/я 765.

Убить Джафара нелегко!

В. Водолазкий

«Крылатый конь несет меня
Туда, где я и в мыслях не был...»

Народная песня

Постоянные читатели нашего журнала помнят, с чего начиналась наша рубрика «Игры»: в третьем номере «Монитора» за 1992 год мы напечатали статью господина Лукашука «Ты записался в убийцы Джафара?» Мы надеялись, что эта публикация поможет отечественным пользователям и программистам справиться с коварным визирем и положит конец бесчинству басмачей на экранах российских мониторов. Увы, нашим надеждам не суждено было сбыться. Фирма Broderbund Software выпустила новую версию игры Prince of Persia 2.0, которая, несмотря на свой огромный размер (6.5 МБайтов) и поддержку только VGA монитора, стремительно распространяется в отечественном дисковом пространстве.

Но перед тем как приступить к изложению самой игры, позвольте поделиться некоторыми соображениями о жанре. Я постараюсь не повторять материал, уже публиковавшийся на страницах нашего журнала, а сказать что-нибудь новое. Как правило, все восхищаются превосходной графикой и музыкальным оформлением игры (к сожалению, платы типа Adlib и Sound Blaster, не говоря уже о Roland, являются пока экзотикой, причем весьма дорогостоящей). Но никому не приходит в голову, что создание игры объемом 6.5 МБайтов возможно только в том случае, если тщательно продумана ИДЕЯ ИГРЫ, разработана методическая база для построения сложных сценариев и определена схема взаимодействия различных персонажей в ходе игры. Prince of Persia, как правило, относят к лабиринтным играм, сравнивая ее с классическими аркадами типа Lode Runner. Но ведь это не так! Prince of Persia относится к одному из самых популярных во всем мире классу игр Dungeons&Dragons. Другим примером может служить серия Castle of Wolfenstein (Бей Гитлера!). Игры этой серии издаются огромными тиражами не только в виде компьютерных программ, но и в виде книг-игр, и в виде настольных игр с обычными кубиками и фишками. И при всей своей простоте пользуются огромной популярностью.

К чему все это говорится? А вот к чему. У нас в России (простите меня, программисты из соседних стран СНГ, я просто ничего о вас не знаю) уже появилось несколько фирм по программированию, разрабатывающих достаточно качественные игры, использующие возможности видеоадаптеров и звуковых плат. Мне довелось общаться со специалистами из фирмы «НИКИТА», Института новых технологий образования, ТОО «Ижица» и Maddox Games. Кое-что из их разработок мы поместили в наш компьютерный ларек. Но, за исключением фирмы «Ижица», предложившей игру «Бизнесмен» (созданную, по всей видимости, под влиянием игры RailRoad Tycoon фирмы Microprose), основное внимание уделяется разработке концептуально простых игр аркадного или логического типов. Но о них мы поговорим отдельно.

А пока я хочу привлечь внимание к пустой нише на рынке программных средств России: приключенческим играм. Мы постараемся по мере наших возможностей осветить основные математические методы и алгоритмы, используемые при создании подобных игр. Но мы надеемся и на помощь наших читателей.

Читатель уже, пожалуй, устал от моих рассуждений. Ну что же, пора вернуться к теме статьи.

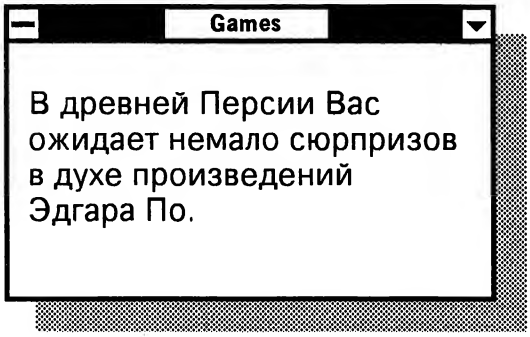
Итак, что нового в древней Персии? Счастье победителя оказалось недолговечным. Через несколько дней после свадьбы Ваш герой направляется в диван, где султан вершит справедливый суд над своими подданными. Перед дверями Вас пронизывает сквозняк, подобный тому, который Вы ощущали, когда проходили через магическое зеркало на четвертом уровне первой серии игры. Полагая, что все это суеверия, Вы храбро открываете дверь и входите внутрь. Но что это? Все присутствующие с удивлением и недоумением пристально вглядываются в Ваше лицо! Подойдя ближе к принцессе, Вы видите, что она удивлена, так же как и остальные. На Ваш естественный вопрос, что же произошло, из тени выступает знакомая Вам фигура, в которой Вы узнаете самого себя! На мгновение черты лица смазываются (напоминая фильм Night Walkers, снятый по роману Стивена Кинга), и Вы узнаете в своем двойнике Джафара, колдовским образом спасшегося от гибели. Вы же оказываетесь одеты в нищенские одежды, и Ваше единственное спасение от гнева самозванца — бегство! Прыгая в окно, Вы оказываетесь на крыше дворца, а затем...

А затем сюжетная линия оказывается в Ваших руках. Но прежде всего отметим изменения в клавишах управления, которые поначалу покажутся Вам неудобными. Теперь для того, чтобы выхватить саблю, необходимо нажать клавишу <Ctrl>, а чтобы ее спрятать — клавишу <5> на цифровой клавиатуре или стрелку вниз. Не спрятав саблю, Вы не сможете бежать или прыгать.

Итак, Вы выскочили из окна. Сразу же выхватывайте оружие. Герой сам развернется навстречу опасности (это правило не работает, если противник нападает сразу с двух сторон!). Поразив врага, прячьте саблю и бегите навстречу приключениям. На первом уровне Ваша задача состоит в том, чтобы покинуть город на ожидающем Вас корабле. Поэтому, дойдя до «края мира», прыгайте вниз и бегите на пристань («деревянные мостки»), не ввязываясь в драку. Противник будет возникать снова и снова, но судно Вас ждать не будет. Вам надо торопиться.

Попав на корабль, Вы можете передохнуть. Но, к сожалению, недолго. Ибо Джафар, узнав о том, что Вам удалось избежать «справедливого возмездия», насылает на Ваш корабль бурю, которая разбивает в щепки судно и выбрасывает Вас на пустынный остров. Обратите внимание, что империалисты в погоне за прибылями не пощадили ни одного из членов экипажа корабля, бескорыстно помогших Вам. Из этого можно сделать единственный вывод: авторы игры тайно сочувствуют Джафару и постарались приложить максимум усилий к тому, чтобы Вы не смогли предъявить свои права на персидский престол. Так что мужайтесь, Вам предстоит изрядно попотеть!

Прежде всего Вам надо найти выход с острова. Как показали проведенные нами исследования, Вам для этого необходимо отыскать вход. В данном случае — вход в пещеру. Это несложно, подсказка необходима только одна: чтобы суметь открыть входную дверь, необходимо наступить на все ступеньки, КРОМЕ отмеченной тайным знаком.



В пещере Вы сможете познакомиться с новинками человеконенавистнической технологии эпохи Синдбада-Морехода. Это шипы, которые более не вырастают из-под земли, а располагаются на вертикальных стенах, что позволяет Вам при случае пришпилить противника или наоборот, поскольку это дело обоюдное. Во-вторых, это мортиры, срабатывающие от нажатия на кнопку. Некоторые из них можно обмануть, если передвигаться через кнопку на корточках. Появились также лужи с противно чавкающей протоплазмой, которые уничтожают с первого попадания, и скелеты, которые Вы можете вывести из строя, хотя и ненадолго. Но не забывайте, что Ваша задача состоит не в геноциде безобидных обитателей подземных застенков Джафара, а в спасении порабощенной страны. Между прочим, после Вашего бегства принцесса тяжело заболела, и, если Вы не успеете вовремя (оставшееся время можно увидеть, нажав на клавишу пробела), она умрет. Поэтому, если за Вами гонится толпа скелетов, не проявляйте чудеса героизма. Вам надо быстрее уносить ноги, а там все как-нибудь образуется.

Наконец, в одной их пещер Вы обнаружите симпатичный темно-синий ковер. Вы вполне логично подозреваете, что это ковер-самолет, и это, вообще говоря, правильно. Но не спешите удобно располагаться на нем. У Вас есть все шансы превратиться в массу, подобную Whiskas, ведь решетка на верхнем своде пещеры закрыта, иначе атмосферные осадки неизбежно повредят тонкое оборудование летательного аппарата. Обратите внимание на скромную педаль, размещенную на верхней галерее. Именно она — Ваш ключ к спасению. Но добраться к ней окажется нелегко. Путь караулит «неубиваемый» скелет, охраняющий висячий мост. Если Вы попытаетесь ввязаться в драку, древнее прогнившее сооружение не выдержит, и Вы вместе со своим противником рухнете вниз на радость Джафару. Кроме того, кнопка, открывающая дверь, находится в начале моста, а сама дверь — за мостом, скелетом и кнопкой с мортирой. Расстояние подобрано таким образом, что преодолеть его можно только непрерывными прыжками, пролетая мимо скелета и не наступая на кнопку мортиры. Вам придется некоторое время потренироваться, но другого пути просто нет.

Новая версия игры стала менее прямолинейной, чем первая. Иногда оказывается, что выход с уровня находится совсем рядом: чтобы дойти до него, нужно всего лишь пятнадцать — двадцать секунд. Может быть, поэтому заветное слово MEGAHIT в новой версии не работает.

Но вернемся к нашему герою. Ковер-самолет, выполняя прямой рейс «Пещеры — Развалины», переносит Вас ко входу в разрушенный замок. Жуткое, я Вам доложу, место. Во-первых, это змеи. Разят насмерть с одного укуса. Но при известных навыках одним ударом уничтожаются. Во-вторых, медузы. Противно пищат и очень эффективно дерутся. Для борьбы с ними тоже необходимо попрактиковаться. Но это живые обитатели. В развалинах Вас ненавидят еще сильнее, чем в пещерах. На сцену выступает исключительно эффективное средство против любителей играть в рабочее время, получившее у последних ласковое название «свинорез». Что это такое — увидите сами. Спутать его ни с чем невозможно! И учтите, завидя «свинорез», Ваш герой сразу вспомнит, что можно не только бегать и прыгать, но и ползать.

На этом уровне Вы можете найти среди проломов в стене трубы, поползав в которых, Вы сумеете пополнить запас жизненных сил. Но будьте осторожны! Змеи в трубах встречаются достаточно часто, но мечом Вы воспользоваться не сможете. Выход в этой ситуации один — отползать назад до выхода из трубы, а затем разделаться с пресмыкающимся ловким ударом.

В ходе Ваших блужданий Вы обнаружите меч-кладенец, бесхозно валяющийся на полу. Не стесняйтесь, возьмите его. Но если у Вас осталось меньше трех «литров» жизни, Вам конец. Ибо потрясение окажется слишком велико. Герой теряет сознание, и перед ним проходит картина далекого прошлого, из которого становится известно... О ужас! Это же родной дом Вашего героя, и он вовсе не бездомный нищий, а настоящий принц! Но злой враг Джафар разрушил замок, и принц остался единственным, кто может отомстить за гибель родных и всего своего народа.

А для этого надо добраться до злодея. Тут в этом Вам поможет волшебный конь, мирно стоящий в виде скульптуры на одном из следующих уровней. Чтобы попасть на него, Вам придется вначале перебить всех медуз, пройдя понизу, а затем быстро пробежаться по верхней галерее и прыгнуть прямо на коня. Тогда волшебное животное оживет и перенесет Вас к храму Бога Гора, в котором окопались враждебно настроенные религиозные фанатики.

Это не менее ужасное место. Прежде всего, Вашим противником будут не «старушки», как это покажется по их внешнему виду, а хорошо обученные басмачи в разукрашенных перьями шлемах. Кроме того, в дополнение к «свинорезу» появляются «соковыжималки» (знакомые по фильму «Призрак замка Моррисвиль») и «перевертыши» — педали, поворачивающиеся под ногами и сбрасывающие Вас на этаж или два ниже, а то и прямо на «свинорез».

В храме Вас ожидает немало сюрпризов в духе произведений Эдгара По. Рекомендация может быть только одна: сначала думайте, потом делайте. Кое-где придется проходить и сквозь «соковыжималку» (хотя это уже в духе братьев Стругацких).

Но враг уже поджидает Вашего героя! Последний этап самый сложный. И не только с точки зрения возросшей квалификации противника. Прежде всего необходимо догадаться, что нужно сделать. А это окажется нелегко. Надо признать, что завершающий уровень больше напоминает игры фирмы Sierra On-line, в которых надо сначала понять, что от тебя требуется по условиям игры. А дальше, как говорится, дело техники.

Все же не могу удержаться от нескольких подсказок. Но хочется сформулировать их таким образом, чтобы они помогли думающему игроку. Поэтому рекомендации мы зашифруем в виде «афоризмов», пусть и не столь крылатых как у Козьмы Пруткова. Как говорится, «умный догадается, а ...»

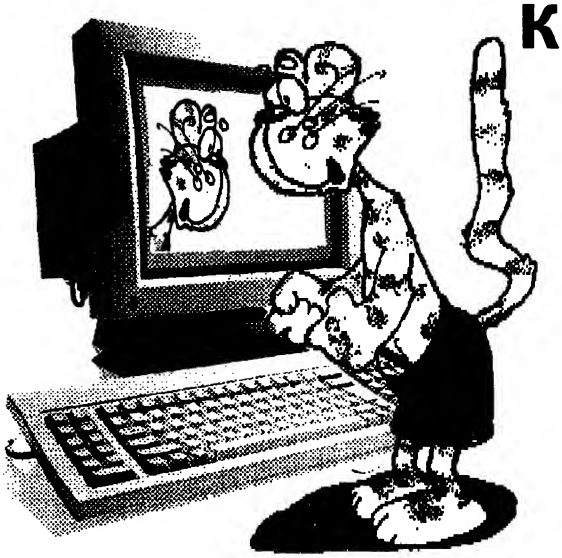
Итак...

«Если фигура на шахматной доске подмигивает, значит, это кому-нибудь нужно...»
«Паучья лапа — путь к совершенству»
«Бей черно-желтых до пятнадцатого стакана»
«Центральный двойник — точка вращения»
«У Джафара слух неважный — подкрадывайся на цыпочках»
«Зато он хорошо бегает»
«Если ты горишь от ненависти — нажми <Ctrl>»

Если Вы сумеете воспользоваться нашей информацией, то сможете разделить радость Вашего героя и... убедиться, что на этом приключения персидского принца не заканчиваются! Что же, мы готовы к встрече с третьей частью. А Вы? ■

Пристальный взгляд на компьютерные игры

Д. Чикин



– Что же из этого следует? — Следует шить. Шить сарафаны и платья из легкого ситца.
– Вы полагаете, все это будет носиться?
– Я полагаю, что все это следует шить.

Ю. Левитанский. Диалог у новогодней елки

Среди множества программ, представленных на стендах международных выставок, рекламируемых в печати или хотя бы просто обсуждаемых в отечественной компьютерной периодике, Вы не встретите, пожалуй, лишь одного класса программ. Программ самых популярных и общеизвестных — компьютерных игр. Хотя игры — первое, с чем знакомится большинство пользователей ЭВМ*, происхождение игр покрыто завесой тайны. И сейчас мы попробуем ее приподнять.

Кто пишет игры?

А кто их не пишет? Студенты и инженеры, рабочие и служащие — был бы под рукой компьютер и свободное время. Если Вы хотите поупражняться в программировании, но у Вас нет конкретной задачи, то игры представляют собой отличное упражнение для программистов любого класса. Остается только жалеть, что у нас в стране не выходят книги под такими названиями, как «Сто игр на BASIC для вашего Apple».

Однако если ограничить множество программистов теми, кто принимал участие в создании коммерческих игр, доведенных до стадии программного продукта, целенаправленно тиражируемого в коммерческих целях, окажется, что одиночек среди них практически нет (как практически нет и отечественных игр). В рассмотрении останутся более или, чаще, менее известные фирмы, посвятившие себя этому странному занятию или сделавшие его одним из направлений своей деятельности.

Если говорить об отечественных создателях игр, то это, главным образом, небольшие группы энтузиастов, помимо программистов, включающие в себя художников и, бывает, композиторов, надеющихся найти таким образом для себя «экологическую нишу» на рынке программных продуктов. А поскольку рынок тот вместе со своими нишами расположен в основном за рубежом, в мире СКВ, то и продукция этих фирм, и их имена часто не известны у себя на родине. Известно же точно только то, что их немного.

Впрочем, коммерческая сторона вопроса заслуживает отдельного рассмотрения, а мы перейдем к разговору о том, как пишут игры.

Анатомия игры

Для начала разберемся, что же такое игра. Не мудрствуя лукаво, объявим сразу, что компьютерная игра — это программа, основной целью которой является развлечение пользователя, то бишь играющего (и тем самым мы отмежуемся от родственного, но имеющего свою специфику класса программ — обучающих).

Дав такое определение, попытаемся заглянуть внутрь игры: что в ней такого есть? С точки зрения восприятия игры, в ней есть три компонента: оформление, сюжет и игровая ситуация.

Оформление — это верхний слой игры, то, что Вы замечаете, взглянув на монитор, за которым кто-то играет: совокупность рисунков, мультипликации, звуков и так далее.

Сюжет — это средний слой, то, что вы понимаете, разобравшись в игре. Или иначе то, что можно описать словами: правила, последовательность действий, закономерности.

Игровая ситуация — ядро игры, совокупность факторов, заставляющая Вас играть, тот абстрактный

* С этого момента автору не удастся скрыть, что сам он пишет игры только для IBM PC, и поэтому все, что здесь написано, к ZX Spectrum или ЕС-1045 не имеет никакого отношения: разве пользователи ЕС-1045 знают, что такое компьютерная игра?!

мир, в котором Вы живете, когда полностью захвачены игрой.

Если это выглядит непонятно, вспомним, например, изумрудики в классической старой игре Digger. То, что эти комочки зеленые и звучат при «съедении», — часть оформления игры. То, что их надо собирать один за другим, добиваясь полной звуковой гаммы, чтобы получить больше очков, — элемент сюжета. А Ваше настойчивое желание отыскать оптимальную последовательность их сбора — результат продуманной авторами игровой ситуации.

Итак, какова же роль этих компонентов в игре? Конечно, важны все три, и уж хотя бы два из трех должны быть идеальны, чтобы игра имела успех. Однако вариантов довольно много.

Интерес играющего к оформлению игры со временем, по-видимому, ослабевает по экспоненте, в силу чего качество оформления мало влияет на долговечность успеха игры, но зато является едва ли не критическим фактором при ее рекламировании и распространении. Современное оформление игр, особенно с появлением мониторов VGA, достигло значительного художественного уровня, и примитивно оформленная игра может пользоваться успехом только в исключительных случаях, подобных легендарному Тетрису. Зачастую оформление может даже иметь решающее значение в игре, о чем мы еще поговорим.

Сюжет как таковой, напротив, редко влияет существенно на восприятие игры. С одной стороны, он воспринимается как основа для оформления: в Wolfenstein 3-D фирмы ID Software игра проходит в фашистском «бункере», в Wing Commander в соответствии с сюжетом Вам показывают мультипликационные сцены в штабе звездных сил и так далее. С другой стороны, игровая ситуация возникает в рамках сюжета, в котором должны быть продуманы число и сложность уровней, характер препятствий, метод начисления очков и так далее. Номинальная же составляющая сюжета не столь важна; в самом деле, разве существенно, что в Digger противники именуются хоббинами и ноббинами, а герой «Перестройки» назван демократом?* Хотя и здесь хватает исключений: например, серии Space Quest, Police Quest фирмы Sierra Online немыслимы без содержательной части сюжета.

Наконец, игровая ситуация — это то, что требует самой детальной проработки, поскольку именно она заставляет игрока снова и снова часами пытаться победить компьютер или своих коллег по работе — будь то игра в шахматы или Xonix. Поэтому создать удачную игровую ситуацию — это, конечно, искусство. Сместить акцент в игре на сюжет, как в уже упомянутых играх серии Quest, требует огромных усилий, а пытаться компенсировать бедность игровой ситуации оформлением почти безнадежно.

Как пишутся игры

Игры сначала действительно пишутся — на бумаге, подобно созданию любого другого программного продукта: появление идеи, разработка технического задания (сценария), программирование пробного варианта, доводка, тестирование и сопровождение.

Конечно, наиболее ответственным этапом является поиск и продумывание самой идеи игры: неудачную идею в хорошую игру воплотить невозможно. На этой стадии не бывает готовых рецептов. Отчасти помогает знание уже существующих игр, отчасти — обсуждение с коллегами и друзьями, но в первую очередь — интуиция. Также трудно сказать, что бывает отправной точкой в создании игры: сюжет, элементы игровой ситуации или, может быть, характерные черты оформления. Например, «Перестройка» родилась из образа лягушки, прыгающей по тонущим под ногами болотным кочкам.

Здесь самое время поговорить о том, как удается сделать игру интересной, за счет чего игра становится игрой.

В первую очередь, как мы уже отметили, играющий воспринимает оформление игры; этот очевидный факт мы далее комментировать не будем.

В следующий момент надо, как минимум, заставить играющего начать играть. Для этого, как правило, в сюжет включается некая цель игры. Иногда про цель просто сообщают: есть, мол, заколдованная принцесса, и надо ее найти. Подразумевается, что после этого играющий начнет искать эту принцессу. Иногда считается, что цель достаточно ясна: если это лабиринт, то надо искать выход, если написано, что «пробел - пушка», то надо во всех стрелять, и так далее. В любом случае можно считать, что цель играющего — поиграть, и подольше. Тогда достаточно поставить его перед неизбежностью действий: запустив Тетрис, играющий обнаружит, что, если не управлять фигурками, игра тут же кончится, и целью становится продержаться как можно дольше. Заодно оказывается, что от этого зависит число полученных очков, и тогда игрок понимает, что интересно бороться за максимальный счет.

Тут мы подходим к центральной проблеме — чем должен быть интересен сам процесс игры? Ответ в общем виде может выглядеть так: должна быть создана такая игровая ситуация, в которой играющий ощущает наличие определенной цели, которой он может достичь, приложив усилия по преодолению некоторых препятствий. Заметим, что целью может являться и просто удовлетворение любопытства — а что меня ждет или кто нарисован на следующем уровне — и тогда основная роль может принадлежать оформлению игры. Под усилиями же можно понимать и решение логических задач, как в шахматах.

Таким образом, продумать идею игры значит, вжившись в образ играющего, детально представить себе его цели в игре и создать такую ситуацию, когда он стремится к ним и постоянно чувствует, что он вот-вот победит компьютер, и дать ему это сделать.

Если в игре все это присутствует, тогда остальное, как говорится, — дело техники, о которой теперь и пойдет речь.

Как игры программируют

В целом программирование игр ничем особым не отличается от программирования многих других задач: используются самые различные типы данных, возникают во множестве вспомогательные алгоритмы типа бы-

* Тут автор не без ложной скромности замечает, что он участвовал в создании «Перестройки» — той самой, которую открывает лик М.С.Горбачева в проломе кремлевской стены.

строго поиска, сортировки, динамического распределения памяти, рекурсивных функций и тому подобных.

Однако характерной чертой практически всех игр является графическое и звуковое оформление, для создания которого требуется разработка нестандартных средств низкого уровня.

Именно в силу этой причины программистам-одиночкам, даже обладающим незаурядной изобретательностью и художественными способностями, обычно не под силу создать коммерческую игру. Эти средства поддержки оформления каждая фирма разрабатывает, как правило, самостоятельно, и, конечно, они являются коммерческой тайной.

Для примера рассмотрим инструментальные средства, которые применяют при написании игр автор статьи и его коллеги по фирме «НИКИТА». Программисты фирмы используют языки С и ассемблер. Собственные инструментальные средства включают в себя несколько утилит и библиотек для обеспечения графических и звуковых эффектов и общесистемной поддержки.

Основной объем инструментальных средств, конечно, связан с графикой. Сразу следует заметить, что подавляющее большинство компьютерных игр для IBM PC проектируется в графике низкого разрешения: 320 на 200 точек. Причина этого, в первую очередь, высокая скорость обработки и вывода изображений: многие графические функции связаны с поточечной обработкой изображений, и, следовательно, именно разрешение определяет скорость их работы. Другой причиной служит возможность запуска игры на любом видеоадаптере с использованием лишь изображений, подготовленных для VGA: для других адаптеров требуется лишь перекодировка цветов. Кроме графики низкого разрешения, инструментальные средства фирмы поддерживают и ряд форматов графики высокого разрешения, которые, однако, по описанным выше причинам редко используются.

Games

Характерной чертой всех игр является графическое и звуковое оформление, для создания которого требуется разработка нестандартных средств нижнего уровня.

Итак, как же создается графическое оформление? Сначала художник (или сам программист) готовит с помощью какого-либо из графических редакторов — PC Paintbrush, Autodesk Animator, DeluxePaint Animation — все требуемые изображения: заставки, фазы мультфильмов и так далее. Все они затем переводятся в один графический формат, в качестве которого выбран наиболее простой PCX формат. Затем специальная утилита переводит полученные изображения в один из разработанных фирмой форматов хранения изображения. Эти форматы напоминают принятые в Windows: в заголовке хранится информация о типе формата, длине файла, размерах изображения, наличии прозрачного цвета, а после заголовка следует изображение либо в виде таблицы кодов цвета пикселов, либо — для простых рисунков — в виде последовательности пар {цвет, число повторяющихся точек этого цвета}. Эта же утилита создает и файл, в котором отдельно от изображения сохраняется информация о палитре цветов рисунка и записываются оптимальные таблицы перекодировки в палитры с меньшим числом цветов.

Если некоторая последовательность рисунков составляет отдельный мультфильм, то для экономии памяти и ускорения дальнейшего вывода его на экран можно использовать метод градиентных изображений. В этом случае утилита сравнивает содержимое двух PCX файлов, содержащих соседние фазы мультфильма, и записывает на диск лишь разницу между ними — градиент. Очевидно, что вывод на экран последовательности градиентов эквивалентен показу последовательности фаз.

Отдельно готовится все, что требуется для звукового сопровождения: файлы с преобразованным в цифровую форму звучанием инструментов и нужных для оформления шумов и файлы с записью используемых мелодий (они могут состоять из нескольких голосов). Вообще говоря, все логически самостоятельные данные записываются в отдельные файлы: например, массивы символьных строк, используемые в программе, образуют просто ASCII файлы. Полученные несколько десятков или чаще сотен файлов, наконец, собираются в один архив с помощью фирменной утилиты, функционально напоминающей любимые всеми архиваторы типа PKZIP или ARJ: готовый продукт будет состоять из одного исполняемого модуля и архива с данными. Такой подход, помимо прочего, позволяет максимально изолировать данные от кода, что упрощает разработку и сопровождение программы, и скрыть их от конечного пользователя.*

Заметим, что графические функции времени исполнения частично — в наиболее критических местах — написаны в нескольких вариантах: для процессоров 8086, 80286, 80386. Они реализованы в виде специальных драйверов, загружаемых после запуска программы из того же архива, что и другие данные. В результате программа в начале работы определяет характеристики аппаратуры, открывает архив, загружает требуемые драйверы и по ходу работы загружает в память требуемые данные из архива.

Что же входит в состав функций времени исполнения? Подробный перечень был бы слишком длинным, поэтому перечислим лишь основные группы функций: это уже упоминавшиеся функции определения характеристик ЭВМ и функции доступа к архивам данных; графические процедуры объединения планов, масштабирования, поворотов изображений, перекодировки и установки палитры и, конечно, быстрого вывода на экран, чтения с экрана, процедуры работы с графическими шрифтами; вывод звуков заданной высоты и мелодий, функции распределения памяти и управления «мышью» и ряд других. Причем подпрограммы вывода обеспечивают вывод звуков на любой из трех поддерживаемых конечных источни-

* Кому не захочется поэкспериментировать с EXE модулем, если в нем при просмотре видны тексты всех сообщений!?

ков: внутренний динамик, Covox (ЦАП, подключаемый к разъему LPT1 или LPT2) или SoundBlaster.

Особенностью графических функций является то, что они позволяют любые операции с изображениями производить в оперативной памяти, с тем чтобы на экран выводилось лишь готовое изображение. Тем самым программист избавлен от необходимости «подчищать» что-то на экране, надеясь, что это будет почти незаметно.*

Наконец, приведем для примера фрагмент текста программы на языке С, решающей простейшую задачу мультипликации: произвольное движение объекта из N фаз одинакового размера поверх фона. Пусть BUFFER — имя типа «указатель на буфер с изображением», и в программе имеются описания, приведенные в примере 1.

Пример 1

```
BUFFER fon;          /* адрес фонового изображения */
BUFFER phase[N];     /* последовательность фаз объекта; */
BUFFER tmp;          /* временный графический буфер таких
                        же размеров, что и все phase[i] */
int x, y;            /* текущие координаты фазы */
int i = 0;           /* номер текущей фазы */
long ticks;          /* текущее значение таймера BIOS */
```

Тогда, не вдаваясь в подробности реализации начала и конца движения и вычисления координат фаз, один цикл движения можно было бы записать так (пример 2).

Пример 2

```
SetOrigin(tmp, x, y);      /* tmp расположили на месте
                              предыдущей фазы */
x = ...; y = ...;          /* вычислили положение новой фазы */
SetOrigin(phase[i], x, y); /* изменили логические координаты
                              новой фазы */
CutPlane(fon, tmp);        /* скопировали в tmp находящийся
                              на его месте фон */
Merge(phase, tmp);         /* и поверх него - часть новой фазы,
                              попадающую на место предыдущей */
while( Ticks()-ticks<T );  /* задержка в T*0,182 сек. между фазами */
ticks = Ticks();           /* в TurboC, например,
                                 long Ticks (void) {biostime(0,0);} */
PutImage(tmp);             /* стерли предыдущую фазу;
                              возможно, вывели часть новой */
PutImage(phase[i]);        /* вывели новую фазу целиком */
if ( ++i == N ) i = 0;     /* перешли к новой фазе */
```

Тут, пожалуй, у читателя уже может родиться вопрос: «При таком-то инструментарии, наверно, и написать игру можно за пару дней?» «Нет», — ответит автор, — «самая простая игра, чтобы стать Игрой, требует не менее одного человеко-месяца. А на солидных западных фирмах крупномасштабные проекты могут потребовать даже десятков человеко-лет труда.»

Как игры продают

Теперь вернемся к коммерческой стороне вопроса; именно от нее в конечном итоге зависит и будущее состояние игрового рынка в нашей стране, и перспективы развития индустрии компьютерных игр. И хотя разговор этот потребует не одной статьи, некоторые замечания хочется все же сделать не откладывая.

Текущее состояние рынка игровых программ для IBM PC как на Западе*, так и у нас, по-видимому, все представляют себе. Оно не принципиально отличается от состояния рынка программных средств вообще, а в нашей стране — и не в лучшую сторону. Но все же появляются обнадеживающие тенденции.

Во-первых, к настоящему времени у нас в стране сформировались коллективы, способные создавать и создающие конкурентоспособные коммерческие игры. Хотя сами эти игры отечественным игрокам зачастую неизвестны, одна из составляющих рынка — предложение, пожалуй, уже есть.

Во-вторых, как ни странно, спрос на игры тоже появляется — не в виде готовности записать их бесплатно у друзей, а официально приобрести за цену продавца. Тому доказательством может служить интерес, который проявлялся к играм фирмы «НИКИТА» на прошедшей выставке SofTool'92 — интерес, несколько неожиданный даже для самих сотрудников фирмы. По-видимому, это связано с ростом парка вычислительной техники вообще, числа компьютеров в домашнем пользовании, в частности, и ростом дифференцированности доходов населения.

В-третьих,... и тут возникает больше вопросов, чем ответов, так как «в-третьих» должна развиваться инфраструктура, связывающая спрос и предложение. А это не только каналы распространения через магазины (книжные?!), компьютерные сети, почту, но и специализированная периодика, и исследования социологов и психологов (кстати, давно ведущиеся на Западе), и связь с органами образования, и, конечно, способы борьбы за свои авторские права, что уже дает неисчерпаемую тему для дискуссий.

И все же: «Что же из этого следует?» Наверно, наступило время активно формировать этот рынок. По крайней мере, игры фирмы «НИКИТА» отечественные игроки увидят в продаже уже в начале 1993 года.

На этом мы закончим рассказ о том, что можно увидеть за парой файлов в вашей директории GAMES: рождение идеи, шлифовку замысла, труд художников и программистов, их фантазию и выдумку. ■

* Заметим, что при тактовой частоте в 12 МГц многоголосое звуковое сопровождение может более чем в два раза замедлить выполнение программы, а в такой ситуации рассчитывать на незаметность каких-либо изменений на экране не приходится.

* Напомним, что Запад простирается на запад вплоть до Японии, Кореи, а также и в других направлениях.

Монитор — 1994

Подписной купон

1	2	3	4	5	6	7	8

МОНИТОР

Адрес _____

Ф.И.О. или название организации _____

Стоимость подписки на 1 номер журнала "Монитор" — 600 рублей. В 1994 году будет выпущено 8 номеров (годовая подписка — 4800 руб.). Отметьте крестиками те номера журнала, которые Вы хотите получить. Перечислите сумму оплаты (действительна до 31 декабря 1993 г.) на счет НТО "Софт-Москва": р/с 467002 в КБ "Банк развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г. Москве, МФО 201791.

SIG PARADOX

Л. Ливеровский

Цивилизованные формы общения пользователей между собой, а также с представителями фирм-разработчиков получают все большее распространение в нашей стране. Пионером в организации такого рода общения среди пользователей персональных компьютеров был Л.Г.Бунич, еще на заре эпохи всеобщей «персонализации» организовавший в Москве постоянно действующий семинар по системному программированию, который работает по сей день.

С тех пор появилось множество разнообразных ассоциаций, постоянно действующих семинаров и групп пользователей, объединяющих специалистов по вычислительной технике на основах общности решаемых задач, используемых в работе программных средств и так далее.

Не были исключением и пользователи программных продуктов не нуждающейся в дополнительных рекомендациях фирмы Borland International.

В 1991 году при активной поддержке Borland в СНГ была создана ассоциация групп пользователей Borland — БорАГ, объединяющая в настоящее время около тысячи индивидуальных и коллективных членов.

БорАГ является некоммерческой организацией, ставящей своей целью организацию информационного обмена между пользователями и установление двухсторонней связи с фирмой Borland.

БорАГ издает информационный сборник БорИС, проводит совместно с фирмой Borland ежегодный конкурс программных разработок «Борланд-Контест», занимается организацией семинаров и многим другим.

Спектр программных продуктов Borland широк, разнообразны и интересы членов БорАГ. Поэтому в рамках БорАГ ее члены стали формировать группы по интересам.

В октябре 1992 года начала действовать SIG PARADOX (SIG — special interest group — группа по интересам), руководителем которой стал активный член БорАГ Каганович Юрий Григорьевич.

Очень скоро тематика SIG расширилась на все деловые системы Borland, и теперь эта группа включает в себя пользователей таких продуктов, как PARADOX, PARADOX Engine, dBase, Interbase, QuattroPro, ObjectVision и даже SideKick. В центре внимания ее членов находятся вопросы применения этих средств в среде различных локальных сетей.

SIG осуществляет постоянные рабочие контакты с АО «Борланд», которое оказывает поддержку SIG.

С появлением в России процессора баз данных Interbase область интересов группы динамично расширяется, не замыкаясь на ставших де-факто стандартом в нашей стране IBM PC совместимых компьютерах.

Члены группы принимают участие в тестировании новых программных продуктов. Группа приняла активное участие в тестировании локализованной (русифицированной) версии PARADOX for DOS 4.0.

Группа имеет свою BBS, пользоваться которой пока могут все и бесплатно.

Есть небольшой, но хорошо подобранный информационный фонд по тематике SIG, существенную помощь в комплектации которого оказывает АО «Борланд».

На коммерческой основе организуются курсы, как для конечных пользователей, так и для профессиональных разработчиков.

Ежемесячно в SIG проводятся семинары (второй четверг каждого месяца). На семинарах обсуждается опыт использования бизнес-продуктов Borland, рассказывается об оригинальных инструментальных и прикладных разработках.

Выступающие говорят о проблемах, возникших при реализации тех или иных проектов, и путях их решения. На семинарах постоянно бывают иногородние члены SIG и приглашенные специалисты.

Регулярно делаются сообщения о новых программных продуктах фирмы Borland. Состоявшаяся на одном из семинаров презентация PARADOX for WINDOWS прошла за неделю до начала официальных продаж этой новой СУБД в США.

SIG не ограничивается рассмотрением вопросов, связанных с бизнес-продуктами фирмы Borland, и обращает свое внимание на все новое и перспективное, что есть на рынке деловых систем и баз данных.

Например, на одном из семинаров выступил Дмитрий Артемов — главный редактор русского издания журнала пользователей FoxPro «The Pinter FoxPro Letter» с интересным сообщением о системе FoxPro (2.0 и 2.5) и о перспективах ее развития.

Об одном из заседаний SIG, которое было посвящено первым опытам эксплуатации PARADOX 4.0 for DOS и сравнению его с PARADOX 3.5, хочется поговорить особо, так как СУБД PARADOX уверенно набирает сторонников среди пользователей в СНГ, и рассказ об этом наверняка заинтересует читателей журнала.

С прекрасно иллюстрированным средствами представления данных QuattroPro докладом выступил активный участник SIG, заместитель генерального директора страховой компании «Союзник» Андрей Анатольевич Бочаров.

Он рассказал о сравнении скоростных характеристик двух версий PARADOX, которое он провел «на скорую руку» сразу после приобретения версии 4.0.

В своих тестах он использовал таблицу следующей структуры (табл. 1).

Таблица 1

Структура тестовой таблицы

Имя поля	Тип
Num	N
Val	S6

В таблице 300000 записей. Первое поле содержало значение, равное номеру записи, а поле Val — случайную последовательность букв латинского алфавита. Было создано 4 варианта этой таблицы — с ключевым полем Num и без него — для каждой версии PARADOX. Объем одной таблицы около 3 Мб.

Сравнивались скорости работы команд поиска ZOOM и LOCATE на компьютере ISA 386DX 25 Мгц, 4 Мб RAM, 12 мс 120 Мб HDD. Производился поиск записи с определенным значением поля Num.

Результаты сравнения приведены на рис. 1.

Рис. 1

Как видно из графиков, чем большее значение поля Num искали, тем больше это занимало времени. Это не удивительно, так как поле Num содержит значение, равное порядковому номеру записи.

На больших значениях, когда время поиска записи достаточно велико (по сравнению со временем интерпретации тестовой программы на PAL и другими факторами), команды ZOOM и LOCATE работают одинаково быстро и PARADOX 4.0 опережает свою предыдущую версию. (Среднее время выполнения одной команды языка PAL, не связанной с обработкой таблиц, составляет 1000-2000 команд/с.)

При поиске относительно небольших значений время работы команд ZOOM и LOCATE различалось и PARADOX 3.5 немного опережал 4.0. Однако эти данные не очень достоверны из-за того, что время поиска становится сравнимым со скоростью работы тестовой программы и сильно сказываются

другие факторы, такие как работа внутреннего кэша (обрабатываемая часть таблицы может целиком находиться в кэш-буфере) и так далее.

Проведенные замеры скоростей работы команды ASK на таблице без ключа и всех команд вместе на таблице с ключом по той же методике не дали достоверных результатов, так как время поиска записи было около 0,1 с, что с учетом других факторов не превышало уровня «шума».

Результаты, представленные Бочаровым, вызвали неоднозначную реакцию слушателей. Многим было трудно поверить в столь незначительную разницу в быстродействии. Высказывались предположения о некорректности использованных тестов.

Далее А.А.Бочаров представил интересные графики, показывающие влияние доступной оперативной памяти на время работы команды ASK в PARADOX 3.5. Использовалась та же таблица. Искались все записи, содержащие в поле Val букву N (..N..) или две буквы N (..NN..) подряд.

Результаты представлены на рис. 2.

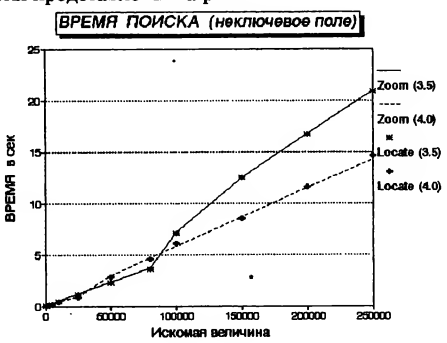


Рис. 2

Он сказал, что ничего удивительного эти графики не содержат, но представляют определенный интерес, так как дают общее представление о влиянии памяти на быстродействие.

Чем больше записей содержит таблица ANSWER, тем больше время поиска, и оно растет линейно. Причем видно, что когда количество обрабатываемых записей и размер таблицы ANSWER переходят определенную границу (все перестает «влезать в память»), PARADOX, видимо, начинает активно использовать подкачку с диска, и время поиска нарастает скачком. Затем влияние этого фактора снижается, и графики идут далее опять почти линейно.

Из этих графиков видно также, что чем больше размер таблицы ANSWER, тем раньше наступает скачок в увеличении времени поиска.

По характеру сближения двух графиков для разной величины доступной памяти можно судить о влиянии количества свободной памяти на скорость работы. Когда число обрабатываемых записей и размер таблицы ANSWER велики, время поиска на памяти в 1 Мб мало отличается от времени поиска на 3 Мб.

Таким образом, увеличение доступной памяти более чем в два раза дает незначительный выигрыш в скорости работы. Есть предположение, что характер этих графиков останется таким же и при большей величине доступной памяти.

Затем с сообщением о сравнении скоростных показателей PARADOX 3.5 и 4.0 выступил сотрудник Внуковского отдельного авиаотряда Алексей Самохин. Он рассказал, что ими было выполнено сравнительное тестирование на быстродействие пакетов PARADOX 3.5 и PARADOX 4.0. Тестирование проводилось на компьютере с процессором i386 SX с частотой 33 Мгц, объемом оперативной памяти 4 Мб, диском со временем доступа 18 мс. Основные тесты выполнялись на реальных данных (архив зарплаты предприятия): таблица из 100000 записей размером 4.3 Мб, структура из 11 полей данных разного типа. Каждая версия работала со своими типами таблиц одинаковой структуры и с одинаковыми данными.

Сетевые тесты выполнялись следующим образом. PARADOX располагался на жестком диске той же станции, что и в локальном тесте, а таблицы на диске файл-сервера. Никакие другие задачи в сети одновременно с тестом не выполнялись (сеть ARCNET, ОС NETWARE V 3.11).

Результаты тестирования приведены в таблице 2.

Обобщая данные таблицы, Самохин сказал, что преимущество пакета V 4.0 тем ощутимее, чем сложнее запрос к базе данных. Наоборот, при интерактивной работе с частой сменой форм представления данных на экране скорость работы пакетов примерно одинакова. При работе в сети преимущество новой версии есть, но не столь существенное, как в локальном варианте. Пакет работал в сети устойчиво, сбоев не было.

После докладов состоялось их обсуждение, в ходе которого участники семинара пришли к единодушному мнению, что преимущества PARADOX 4.0 в наибольшей степени, видимо, сказываются на сложных запросах к базам данных.

Колоссального увеличения быстродействия (до 10 раз по сравнению с PARADOX 3.5, как было объявлено официально), вероятно, можно достичь при использовании сложных многотабличных запросов на мощных (по нашим меркам) ПЭВМ — AT 386 с 4 Мб и более RAM.

Однако если прикладная задача работает с тривиальными таблицами и запросами (как в тестах А.А.Бочарова), его использование может не дать заметного увеличения быстродействия. Минимальные требования к оборудованию для работы PARADOX 4.0, объявленные фирмой Borland (AT 286 с 2 Мб RAM), действительно являются минимальными. На таком компьютере PARADOX 4.0 почти всегда работает медленней своей предыдущей версии.

Если же вычислительные ресурсы компьютера невелики (AT 286 с 1 Мб RAM), то использование PARADOX 3.5 для пользователей таких ПЭВМ остается единственным выбором.

На этом точку в сравнении двух версий СУБД PARADOX ставить рано. Мы приглашаем всех заинтересованных лиц не только принять участие в сравнении быстродействия этих двух систем, но и поделиться соображениями об оптимальном выборе инструментария для построения тех или иных приложений, базирующихся на СУБД, как в прессе, так и на заседаниях SIG PARADOX.

Мы хотим верить, что наша встреча на страницах журнала была полезной для читателей и что она будет не последней. ■

Информация для контактов

Телефоны SIG PARADOX: 366-49-51, 366-49-51

Таблица 2

Сравнение скоростных характеристик PARADOX 4.0 и 3.5

Выполняемая операция	Число обрабатываемых записей (Answer)	Время выполнения операции		Коэффициент ускорения
		V 3.5	V 4.0	
I. Локальный вариант				
1. Запрос по 3 полям	7	1 мин. 2 с	27 с	2.3
2. Запрос по 4 полям	5845	1 мин. 26 с	30с	2.8
3. Запрос по 3 полям	47258	4 мин. 25 с	47 с	5.6
4. Сканирование таблицы с обновлением данных	25000	6 мин. 57 с	3 мин. 05 с	2.2
5. RESTRUCTURE table	100000	15 мин. 45 с	1 мин. 40 с	9.45
II. Сетевой вариант				
6. Запрос по 3 полям	47258	2 мин. 45 с	2 мин. 01 с	1.3
7. Сортировка таблицы	100000	11 мин. 05 с	5 мин. 52 с	1.9
8. Слияние таблиц (операция ADD)	75000+25000	8 мин. 07 с	4 мин. 02 с	2.01
9. Сканирование таблицы с обновлением данных	10000	13 мин. 57 с	6 мин. 35 с	2.12
10. Запрос по 3 таблицам	122088	16 мин. 04 с	3 мин. 31 с	4.57
11. RESTRUCTURE table	100000	35 мин. 45 с	15 мин. 11 с	2.32

ПЕРСПЕКТИВНЫЕ ТЕХНОЛОГИИ

Paradox for Windows

Мощная реализация реляционной модели

Paradox For Windows обеспечивает логическую целостность хранимых в базе данных, а также позволяет держать их на диске в зашифрованном виде. Новые типы полей - мемо, OLE, BLOB позволяют применять графический запрос по образцу. Визуализация данных во всех режимах работы: при проектировании схемы базы данных (Linking Export); при создании форм, запросов и отчетов (Form, Report & Query Exports); изображение структуры объектов программы (Object Tree). Все это позволяет создавать структурно сложные базы данных и большие программные комплексы.
Драйвер русификации поставляется бесплатно.
Paradox For Windows - инструмент, с которым сможет работать новичок. Программисту он позволит держать под контролем сколь угодно большой проект.
Скидки для дилеров 35-40%.
Paradox For Windows в областях, непривычных для реляционных СУБД. ***Полное использование графики MS Windows:*** как среда Paradox For Windows, так и приложения, создаваемые с его помощью, используют все средства конструирования графических пользовательских интерфейсов.

Lotus cc: Mail

Электронная почта для Вас

cc: Mail - надежная связь между сетями, операционными системами и аппаратными средствами, между головной организацией и ее филиалами, между ***ВАМИ*** и ***ВАШИМИ*** клиентами по обычным телефонным каналам.
cc: Mail объединяет пользователей в системах ***MS DOS, Windows, OS/2, Apple и Unix.***
cc: Mail - развитая система посылки и приема сообщений. Регистрация, оповещение, распечатка, архивация, запись, уничтожение и многие другие функции обработки сообщений.
cc: Mail - пересылка отдельных документов и пакетов документов. Пакет документов может содержать фрагменты текста, графики, факсимиле, файлы и снимки с экрана.
cc: Mail - расширение числа пользователей до десятков тысяч.
cc:Mail - сохранность передаваемой информации. Уникальные функции управления передачей и приемом информации отслеживают сетевые и аппаратные ошибки, предупреждая пользователя об их появлении.
cc:Mail - защита информации. Шифрование передаваемой информации и дешифрация принимаемой информации только по ключу пользователя. Средства защиты информации сертифицированы ***CIA и US Navy.***
cc:Mail - взаимодействие с другими почтовыми отделениями в пределах одной локальной вычислительной сети и вне ее.
cc:Mail - взаимодействие с удаленными абонентами.
cc:Mail - взаимодействие с другими системами электронных почт: **NOVELL MHS,** ***MCI Mail, PROFS, EZlink, Telelink, Unix Mail/uucp (relcom) и SMTP.***

Москва, Звенигородское шоссе, д. 9 (м. «Улица 1905г.).
Факс: (095) 259-27-27. Тел.: (095) 256-42-42, 256-40-30.

Прислушайтесь к мнению ведущих мировых авторитетов о компьютерах фирмы

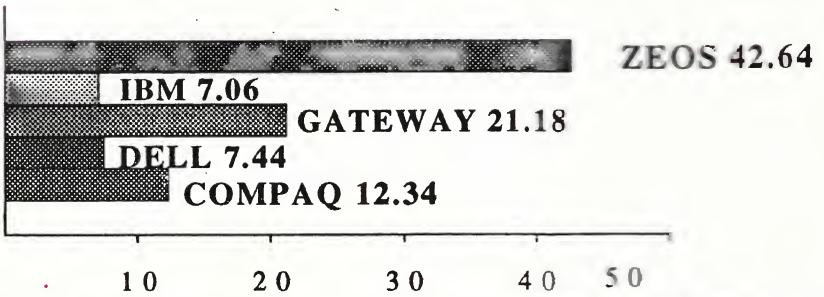
ZEOS®

INTERNATIONAL, LTD.

Вот самые последние оценки

ИТОГИ BENCHMARK-ТЕСТИРОВАНИЯ "GRAPHICS WINMARK" ЛАБОРАТОРИИ ЖУРНАЛА PC MAGAZINE, 1993 Г.

(МЕГАПИКСЕЛЕЙ В СЕКУНДУ)



PC MAGAZINE EDITORS CHOICE

"Zeos... просто смел всех своих конкурентов, в некоторых случаях показывая производительность в десять раз превышающую производительность своих конкурентов.

- PC Magazine, 26 января 1993 г.

"Компьютер 486DX2-66 фирмы Zeos International, благодаря наилучшему сочетанию скорости, высококачественных компонентов и защиты от морального устаревания, буквально "растолкал" всех своих конкурентов..., представляет собой наиболее мощную, с возможностями дальнейшего роста машину по крайне конкурентоспособным ценам.

- Windows Magazine, Май, 1993.

PC WORLD BEST BUY

"Это быстрый, прекрасно спроектированный персональный компьютер с возможностью дальнейшего безграничного увеличения мощи. В результате – награда Best Buy ("Лучшая Покупка).

- PC World, Февраль 1993 г., Best Buy

"Блестящий дизайн материнской платы, великолепная производительность и тщательно продуманные мелочи... Эти качества выделяют ZEOS среди всей толпы".

- PC Computing, Март 1993, Best Value ("Лучшая Ценность")

"Преимущества в технической разработке и качестве... явно выделяют продукцию Zeos среди компьютеров его конкурентов, имеющих схожие цены".

- Forbes (самый авторитетный журнал деловых кругов США), Март 1, 1993

Все поставки в СНГ осуществляются только с заводов фирмы Zeos International, Ltd. (США). Срок поставки стандартных конфигураций компьютеров – 7 дней. Все машины проходят 72–часовое "горячее" тестирование. С каждым компьютером Zeos Вы приобретаете лицензионные копии MS-DOS 6.0 и Windows 3.1 фирмы Microsoft.

Гарантия на все компьютеры – 1 год.

По всем вопросам закупок компьютеров фирмы Zeos International, Ltd. (настольные и серверы на основе процессоров фирмы Intel, начиная с 486SX25 и кончая 486DX2-66; блокнотного формата (цветные и моно VGA) на основе процессоров 386SL и 486SL; palmtop) обращаться в **Международный компьютерный клуб (МКК)** по адресу:
101813 Россия, Москва, Центр, Проезд Серова, 4 (Политехнический музей, подъезд 8, этаж 2, комн. 53, 54).
Тел. (095) 921–09–02; (095) 925–46–67.

2 90 -

Ваша программа, на Вашем компьютере, но гораздо быстрее!

Работу с этими и другими программами ускоряют сопроцессоры Intel

Математические сопроцессоры фирмы Intel:

— **привлекают крупнейших разработчиков программного обеспечения.** Свыше 600 таких компаний во всем мире ориентируются на математические сопроцессоры фирмы Intel, и уже свыше 2100 прикладных программ активно их используют;

— **предлагают проверенное решение для любого компьютера от XT до i486 SX.** Математические сопроцессоры фирмы Intel являются индустриальным стандартом. Во всем мире буквально миллионы пользователей с удовлетворением пожинают плоды повышенной производительности;

— **обеспечивают совместимость.** Поскольку сопроцессоры фирмы Intel разработаны специально для работы совместно с микропроцессорми Intel, на основе которых созданы все персональные компьютеры, они обеспечивают 100%—ную совместимость;

— **означают качество фирмы Intel.** Математические сопроцессоры, изготовленные фирмой Intel, являются наивысшим стандартом в мире и имеют пятилетнюю гарантию;

— **легко устанавливаются.** Надо лишь вставить сопроцессор в отведенное для него гнездо — и готово!

intel®